Detecting Change in an Environment with Wireless Sensor Networks

by

Mingzheng Shi

Submitted in total fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Geomatics
The University of Melbourne
Australia

June, 2010
ABSTRACT

Detecting Change in an Environment with Wireless Sensor Networks

by Mingzheng Shi

Chairperson of Supervisory Committee: Stephan Winter

This thesis is motivated by a new observation tool, called wireless sensor network (WSN). WSN, as any other observation method, must cope with spatial, temporal, and thematic granularities of observations. This thesis investigates how WSNs can efficiently detect different types of changes of spatial phenomena, or objects. Two types of changes, i.e., gradual and abrupt spatial changes, are distinguished based on the quantity of change in particular, WSN inherent granularities. A new spatiotemporal data model is proposed for the representation of dynamic spatial objects in WSNs. An algorithm is then designed for WSNs to detect both gradual and abrupt spatial changes with different spatial, temporal, and thematic granularities. The efficiency of the algorithm is proven by qualitative and quantitative evaluation in WSN simulations. This thesis also proposes a new network structure, called multi-granularity sensor network, for different granularities of observations.
This is to certify that

(i) this thesis comprises only my original work,
(ii) due acknowledgment has been made in the text to all other material used,
(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, appendices and footnotes.

I authorize the Head of the Department of Geomatics to make or have made a copy of this thesis to any person judged to have an acceptable reason for access to the information, i.e., for research, study or instruction.

Signature__________________
Date____________________
PUBLICATIONS

The publications listed below are based on the work presented in this thesis.


ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my advisor and mentor Dr. Stephan Winter for his constant encouragement, guidance and support throughout my PhD research. My sincere thanks are extended also to Dr. Matt Duckham and Dr. Patrick Laube for all the discussions, suggestions, and constructive comments.

I would like to thank my colleagues at the Department of Geomatics, the University of Melbourne, especially to Muhammad Jafar Sadeq, Yunhui Wu, Linjie Guan, Jonathan Arundel for the stimulating discussions and positive criticisms. I appreciate the help of Jonathan Arundel for correcting the grammar of this thesis.

Last but not least, my special thanks go to my family for their moral support.
Contents

1 Introduction 1
  1.1 Problem and Motivation ................................. 1
  1.2 Problem Statement ..................................... 6
  1.3 Approach .............................................. 7
  1.4 Major Results ......................................... 9
  1.5 Organization of the thesis ............................. 9

2 Literature Review 11
  2.1 Spatial Change .......................................... 11
    2.1.1 Ontology of Change ................................ 11
    2.1.2 Representation of Change ............................ 12
  2.2 Wireless Sensor Networks ............................... 14
    2.2.1 Environmental Applications ........................ 14
    2.2.2 In-network Aggregation .............................. 15
  2.3 Environmental Monitoring Using Wireless Sensor Networks ... 20
    2.3.1 Snapshot-based Approach ............................ 20
    2.3.2 Event-based Approach ............................... 21
  2.4 Conclusion ............................................ 23

3 Spatial Change 25
  3.1 Representation of Continuants ............................ 25
  3.2 Representation of Changes ............................... 27
  3.3 Granularity of Observation .............................. 28
  3.4 Abrupt and Gradual Changes ............................ 31
  3.5 Topological and Non-topological Changes ................ 33

4 Spatiotemporal Data Model 35
  4.1 Sensor Network ......................................... 35
    4.1.1 Sensor Network Model ............................... 35
    4.1.2 Traversals ......................................... 37
    4.1.3 Traversal Orientation ............................... 38
4.2 Change Representation in Sensor Network ........................................ 38
  4.2.1 Boundary Edges ............................................................... 38
  4.2.2 Insertion and Deletion of Edges ......................................... 40
  4.2.3 Abrupt and Gradual Changes in WSNs .................................... 41
4.3 Change and Granularity ............................................................. 43
4.4 Multi-Granularity Sensor Networks .............................................. 44
  4.4.1 Tree and Planar Graph ...................................................... 45
  4.4.2 A New Network Structure .................................................. 48

5 Algorithms ......................................................................................... 51
  5.1 The BROADCAST component ..................................................... 52
    5.1.1 Boundary Edge Determination ........................................... 52
    5.1.2 Inserted and Deleted Edges .............................................. 55
  5.2 Triangulation and Planar Graph ................................................ 56
  5.3 Trajectory .................................................................................. 57
    5.3.1 Leader and Trajectory ....................................................... 57
    5.3.2 Trajectory Maintenance ..................................................... 60
  5.4 Change Detection ........................................................................ 62
    5.4.1 Appearance and Disappearance .......................................... 62
    5.4.2 Expansion and Contraction ............................................... 62
    5.4.3 Merge and Split ............................................................... 64
    5.4.4 Self-merge and Partial-split ............................................ 65
    5.4.5 Summary .......................................................................... 67
  5.5 The DETECT Component .......................................................... 67
    5.5.1 Preliminaries ................................................................. 68
    5.5.2 Traversal Approach 1 ....................................................... 72
    5.5.3 Traversal Approach 2 ....................................................... 79
    5.5.4 Detecting Change ............................................................ 81
    5.5.5 Summary of the DETECT Component ................................. 82
  5.6 The UPDATE Component .......................................................... 84
  5.7 Multi-Granularity Sensor Networks .............................................. 87
    5.7.1 Dominating Sets ............................................................. 89
    5.7.2 Number of Parents .......................................................... 89
    5.7.3 Centralized Construction Algorithm ................................... 92
    5.7.4 Decentralized Construction Algorithm ................................. 94
    5.7.5 Simulation Results .......................................................... 98
    5.7.6 Hierarchical Change Detection Algorithm ............................ 99
List of Tables

6.1 The comparison of the message transmission and area approximation for appearances of areal objects. .......................... 106
6.2 Summary of the regression analysis result \( y = a \times x^b \) for the combination of BROADCAST and DETECT components. .... 109
6.3 Summary of the regression analysis result \( y = a \times x^b \) for the UPDATE component. .................................................. 109
6.4 Average messages sent per active node for the DETECT component. ................................................................. 114
6.5 Average messages sent per active node for the UPDATE component. 114
6.6 Sum of average messages sent per active node for BROADCAST, DETECT and UPDATE components. ..................... 116
6.7 The number of nodes that transmit 1, 2, 3, 4, or 5 messages for the eight different types of changes. ......................... 119

7.1 Results of case study. ................................................................. 128
List of Figures

1.1 Spatial granularity is one of the three components of the granularity of change representation. If a phenomenon is observed with a coarser spatial granularity, then the representation of change requires at least one cell to change, as shown in (a)-(b). With a finer spatial granularity of observation, spatial change can be represented in finer granularity, as shown in (c)-(d). 2

1.2 Temporal granularity is one of the three components of the granularity of change representation. (c)-(e) have a finer temporal granularity with a time interval of $\Delta t = t_2 - t_1 = t_3 - t_2$. (a)-(b) have a coarser temporal granularity with a time interval of $2\Delta t$. 3

1.3 Thematic granularity is one of the three components of the granularity of change representation. (a)-(b) have a coarser thematic granularity with two classes: white and gray. (c)-(d) have a finer thematic granularity with three classes: white, gray, and black. 4

1.4 Examples of insertion and deletion of edges. The inserted edges are denoted as thick continuous lines, and the deleted edges are denoted as dashed lines. 8

2.1 An example of tree structure. Node $a$ is the root of the tree, and $e, g, h$ are the leaf nodes. Node $f$ is the parent of node $g$ and $h$. 16

2.2 Examples of cluster structure. (a) Non-overlapping clusters. (b) Overlapping clusters. In (a) and (b), the large dashed-line circles represent one-hop communication ranges and two-hop communication ranges respectively. 18

2.3 An example of planar graph. Different from a tree structure, there are no hierarchical relations among nodes in the graph. 19

3.1 An example of a partition $S$ based on a taxonomy $Y = \{y_1, y_2\}$. 27

3.2 Region collections change over time from $t_1$ to $t_2$. 28
3.3 The region $X$ has been subdivided into 16 cells, and there are 16 nodes in a sensor network. There is a one-to-one correspondence between nodes and cells. Sensor nodes, e.g., $a$, $b$, $c$, and $d$, are represented as small squares.

3.4 Examples of abrupt and gradual changes.

3.5 An example of continuous but not gradual change.

3.6 Different temporal granularities of observations can result in different granularities of spatial changes. The observation of the first row has a coarser temporal granularity than that of the second row.

3.7 Six topological changes and two non-topological changes.

3.8 (a)-(b) The expansion is gradual. (c)-(d) The merge is abrupt.

4.1 Sensor network structure. (a) An example of a communication graph $G$. (b) The set $\text{neighbor}(v)$ of each node $v$ is sorted into clockwise order. (c) The directed edges and cycles of a node $v$.

4.2 Traversals. (a) The traversal of non-cycle edges based on the clockwise rule. (b) A region with a hole can be detected by the traversal orientations.

4.3 An areal object is represented as a sequence of directed edges in a sensor network. Directed edges within an areal object are classified into boundary edges (solid-line arrows) and non-boundary edges (dashed-line arrows), as illustrated in (a). Usually only boundary edges will be illustrated in the figure, as in (b).

4.4 The insertion and deletion of boundary edges. Inserted boundary edges are marked by thick solid-line arrows, and deleted boundary edges are marked by thick dashed-line arrows. Note that only boundary edges are illustrated in the figure.

4.5 Gradual and abrupt changes in WSNs. Sensor nodes that are located in the areal objects are represented as solid squares. (a)-(b) is a gradual change. (c)-(d) is an abrupt change.

4.6 There are relations between granularity of observation, granularity of spatiotemporal data, and granularity of spatial changes.

4.7 (a) and (b) have different temporal granularities of observations, as will be investigated in Sections 5.1 - 5.6.

4.8 (a) and (b) have different spatial granularities of observations, as will be investigated in Section 5.7.

4.9 A centralized structure. There are hierarchical communications between sensor nodes, e.g., $b$, and the sink $a$. 

xiv
4.10 A tree structure. The topology among sensor nodes in (a) can be represented as in (b). Note that (b) is only an abstract representation. .......................................................... 46
4.11 Monitoring areal objects using a tree structure. .................. 47
4.12 Monitoring areal objects using a planar graph structure. ...... 47
4.13 The construction of a new network structure. A subset of nodes in the first level (a) is selected to build the second level (b). .... 48
4.14 The new network structure has the capability of both hierarchical and peer-to-peer communications. ......................... 49

5.1 Boundary edges determination. Sensor nodes located in areal objects are represented as filled squares. Empty squares are the nodes that are not in areal objects. Active nodes are marked by small circles. .................................................. 53
5.2 Edge insertion and deletion. The edges at the node $v$ are highlighted. Inserted and deleted boundary edges are marked by thick solid-line arrows and thick dashed-line arrows respectively. ... 55
5.3 (a)-(b) A triangulation. (c)-(d) A planar graph. .................. 56
5.4 Leaders and leader trajectories need to be updated when areal object changes over time. Leaders are marked by circles, and dashed lines represent the leader trajectories. The sink is marked by a star. ................................................................. 58
5.5 Trajectory Identity. Boundary edges of the areal object have the same identity as the trajectory edges. ............................... 59
5.6 The trajectories need to be updated when areal objects change over time. The star represents the sink, different circles or nodes at one time step represent different leaders, and a solid line represent a path between two nodes. There may be multiple stars at one time step, but these stars represent the same sink at a single location. ................................................................. 61
5.7 Trajectory maintenance for regions with a hole. A region with a hole is specified with superscript $-$, e.g., $B^-$. ................. 61
5.8 Appearance and disappearance. (a) an areal object appears. (b) is an abstract representation of the traversal in (a). (c) an areal object disappears. (d) represents the traversal in (c). Note that the node identities, i.e., $a$ and $b$, is in lower cases, and trajectory identities, i.e., $A$, is in upper cases. ......................... 63
5.9 Expansion and contraction. The traversal for expansion or contraction has two segments, 1 and 2, and two KEY nodes, $b$ and $h$, as in (b) and (d). ................................. 64
5.10 A special case of expansion, in which the traversal only has one
segment. This special case can exist in other types of changes as
well. ................................................................. 64

5.11 Merge and split. The traversals for merge and split include four
segments. Two segments of the traversal are on inserted edges,
and the other two segments are on deleted edges, as in (b) and
(d). The traversals start at node a. .......................... 65

5.12 The traversals for self-merge and partial-split include four seg-
ments and four KEY nodes. ................................. 66

5.13 There are eight different types of basic traversals that can be used
to identify eight different types of changes as in (a)-(h). ......... 67

5.14 Other types of traversals can be considered as the combinations
of the eight basic traversal types. The traversal in (b) is a com-
bination of two traversals in Figure 5.13(a) and (b). ........... 68

5.15 It is possible that there are absent segments in a traversal. KEY
nodes are marked by circles. (a) A segment of deleted edges is
absent at KEY node b. (b) A segment of deleted edges is absent
at KEY node d. (c) Two segments of inserted edges are absent
at KEY nodes b and d. ........................................... 70

5.16 Expansion and contraction. There are two active nodes d and e
in both expansion and contraction. Transition edges are marked
by dashed-line ellipses. ........................................ 72

5.17 Merge and split. There are two active nodes d and v in both
merge and split. The closed trail consists of four KEY nodes and
four segments. .................................................. 73

5.18 An areal object appears between $t_1$ and $t_2$. There are no transi-
tion edges, and thus there are no type 1 node and type 2 nodes. 78

5.19 Traversal Approach 2. The sink is marked by a star. Both of the
two examples are appearances of areal objects. .................. 80

5.20 Updating trajectory identities in the case of appearance, disap-
pearance, expansion, and contraction. In (b), (e), (h), and (k),
the integer numbers in circles illustrate the traversals for updat-
ing trajectory identity. (c), (f), (i), and (l) are the states when
trajectory identities have already been updated by the traversals
in (b), (e), (h), and (k) respectively. ........................ 85

5.21 Updating trajectory identities in the case of merge, split, self-
merge, and partial-split. In (e) and (h), integer numbers in circles
and squares represent two different traversals for updating trajec-
tory identities. (c), (f), (i), and (l), show the updated trajectory
identities. ..............................
5.22 Construction of multi-granularity sensor network. (a) level one graph, (c) level two graph, (e) level three graph. The edges of the graphs at each level are restricted by a unit disk, where the lower level graphs have smaller radius and thus shorter edges, and the upper levels have larger radius and longer edges, as in (b), (d), and (f). 88

5.23 An example of a two-level hierarchical structure. The nodes in the upper level are represented as solid squares. The lower level, with $|V| = 16$, is a planar graph, the upper level, with $|V_p| = 3$, should also form a planar graph. (a) Edges in the lower level. (b) Edges in the upper level are highlighted. 89

5.24 An example of a 2-dominating set. In (b), the set of nodes marked by circles is a 2-dominating set. 90

5.25 Regular Graphs. In the graphs, parents and children are represented as solid circles and empty circles respectively. (a) $\text{deg}(v) = 2$, (b) $\text{deg}(v) = 4$, and (c) $\text{deg}(v) = 6$. The upper level graph $G_p$ has the same degree as the lower level graph $G$, as in (d), (e), and (f). 91

5.26 The first few selections of the centralized construction algorithm. A new parent is selected such that it has the maximum number of common children with the selected parents. In the figure, large dashed-line circles represent the communication range of sensor nodes. Squares are selected parents, solid circles are sensor nodes in the set $\text{child}(V_p)$, and empty circles are nodes not in the set $\text{child}(V_p)$. 93

5.27 The results of the centralized construction algorithm. (a) One of the results of the first two steps of the algorithm, where some of the nodes $v \notin V_p$, represented as empty circles, do not satisfy the requirement of $|\text{parent}(v)| > 1$ (an example is marked by an arrow) (b) The third step of the algorithm makes sure that all nodes not in $V_p$ have more than one parent. (c) A subset of the sensor network. 93

5.28 The summary of 500 results of the centralized construction algorithm. 95

5.29 Initialization phase. (a) The node $v$ received the set of neighbors $\text{neighbor}(v')$ from $v'$ during the initialization (b) The local information stored in the node $v$ after the initialization. 96
5.30 Parent selection phase. (a) In step one, parent $p$ broadcasts a cluster formation message to its neighborhood. (b) In step two, each parent candidate $v$, which is two-hop away from the parent $p$, calculate the common children of $p$ and $v$. (c) The parent candidate $v$, who has the maximum number of common children with $p$, become the new parent in step three. 96

5.31 The results of the decentralized and centralized construction algorithms. The performance of the decentralized algorithm is close to the centralized one, but the centralized algorithm is slightly better. 99

5.32 (a) Total message transmitted of a single simulation run in the decentralized construction algorithm. (b) The impact of network size $|V|$ on the communication cost per node. 99

5.33 Hierarchical structure. (a) A planar graph structure $G$ is the lower level of the hierarchy. (b) The upper level of the hierarchy $G_p$. $G_p$ has smaller node density than $G$. (c) The topological relations among parents (squares) and children (circles) is defined by the unit disk (dashed-line circles). (d) An example of multi-hops communication between two parents. (e) Each parent $p$ has a non-overlap polygon to estimate the area of its neighborhood. (f) Area aggregation by merging polygons. 100

6.1 Experiment of abrupt change (case 3). (a) One-level flat structure, (b) two-levels hierarchical structure. 104

6.2 Quantitative analysis using flat algorithm and hierarchical algorithm. The first and third rows are one-level flat structure. The second and fourth rows are two-level hierarchical structures. 105

6.3 Spatial changes occur at randomized locations within a sensor network. 108

6.4 Total messages sent for the BROADCAST and DETECT components with increasing number of active nodes. 111

6.5 Total messages sent for the UPDATE component with increasing number of active nodes. 112

6.6 Total messages sent for the UPDATE component in the case of contraction. The result in this figure is the same as Figure 6.5(d) but with a different scale. 113
6.7 (a) Snapshots for Case 1.1 (appearance) in Table 6.7. (b) Snapshots for Case 2.1 (disappearance) in Table 6.7. A node with a solid-line circle represents that the node is involved in the traversal of the DETECT component. A node with a slightly larger dashed-line circle represents that the node is involved in the UPDATE traversal.

6.8 (a) Snapshots for Case 3.1 (expansion) in Table 6.7. The different dashed-line circle (with larger gap) at node a represents that node a only receives message in the UPDATE component. (b) Snapshots for Case 4.1 (contraction) in Table 6.7.

6.9 (a) Snapshots for Case 5.1 (merge) in Table 6.7. (b) Snapshots for Case 6.1 (split) in Table 6.7. The smaller and larger solid-line circles represent two DETECT traversals respectively. The largest dash-line circles represent the UPDATE traversal. A thin solid-line circle of a node, e.g., node b in (a), represents that the node only receives a message in one of the DETECT traversals.

6.10 (a) Snapshots for Case 7.1 (self-merge) in Table 6.7. (b) Snapshots for Case 8.1 (partial-split) in Table 6.7.

6.11 An example of expansion.

7.1 The study area, i.e., Sangong river watershed, is located in Xinjiang Uygur Autonomous Region in the Northwest of China [15].

7.2 Case 1. Inserted and deleted edges are marked as thick solid-line edges and thick dashed-line edges respectively. (b) At $t_2$, there are two closed trails, and two expansions have been detected. (c) Similarly, there are other two closed trails for expansions at $t_3$.

7.3 Case 2. (b) At $t_2$, five expansions are detected by nodes a, b, c, d, and e respectively. (c) b, c, d, and e detects expansions at $t_3$.

7.4 Case 3. (b) The sensor network detects three contractions and one split at $t_2$. (c) Four contractions and one disappearance are detected at $t_3$.

7.5 Case 4. (b) Node a detects a contraction, and node b detects an expansion at $t_2$. (c) Node c detects a self-merge at $t_3$.

7.6 Case 5. (b) At $t_2$, node a detects a contraction of the areal object $A$, and node b detects the appearance of a hole $C^-$. (c) Five contractions, one expansion, and one partial-split are detected at $t_3$.

7.7 Case 6. (b) Node a detects a merge of two areal objects.
8.1 Between (a) and (b), there is a merge of three areal objects A, B, and C, and an expansion at areal object A. . . . . . . . . . . . 142

8.2 (a)-(b) Two examples of topological relationships between two areal objects A and B. (c)-(d) Two examples of topological relationships between a line object A and an areal object B. . . . . . . 143

8.3 Changes of topological relationships when spatial objects undergo different types of changes. (a)-(b) Areal object A has expanded (c)-(d) Areal objects A and B has merged into one areal object. . 145
Chapter 1

Introduction

1.1 Problem and Motivation

Many spatial phenomena are continuously changing in time and space [45]. Environmental monitoring, for example, focuses especially on areas of environmental disturbance, such as grazing, fertilizing, pollution, and logging. Environmental disturbance is one of the key driving forces for environmental changes, and different disturbance events can result in diverse changes. For example, a flooding event can result in the increase of soil moisture and the transform from heath land to wetland [93], and a desertification event can cause the decrease of soil moisture and the change from heath land to non-native grassland.

The above examples describe continuous change in the physical world. In contrast, there exist changes in social reality that are regarded as discrete [42, 44, 66], e.g., the split of one land parcel into two parcels due to the change of land parcel ownerships. The distinction of the physical reality and the social reality is included in Frank’s tiers of ontology [42, 43]. This thesis focuses on detecting changes of natural phenomena in the physical reality.

In the tiers of ontology, the physical reality in Tier 0 is observed by, for example, technical instruments. Observations are incomplete, imprecise, approximate, and discrete. For example, an observation with a remote sensing technique can obtain an image of a regular grid of pixels or cells. A regular grid of cells is frequently used as examples in Chapters 1 and 3.

As a result of observations, there is a distinction between the continuous change in physical reality and the discrete representation of change in spatiotemporal databases. This thesis concentrates on the discrete representation of change for natural phenomena. The representation of change has granularities. The granularity of change representation is a result of spatial, temporal, and thematic granularities of observations. The granularity of change repre-
Figure 1.1: Spatial granularity is one of the three components of the granularity of change representation. If a phenomenon is observed with a coarser spatial granularity, then the representation of change requires at least one cell to change, as shown in (a)-(b). With a finer spatial granularity of observation, spatial change can be represented in finer granularity, as shown in (c)-(d).

Presentation is a discretization and quantification of the continuous change in the physical world. Three examples are provided to explain the spatial, temporal, and thematic components of granular spatial change.

Firstly, suppose a logging event reduces forest coverage by 1 km$^2$. If the change of forest coverage is observed in a finer spatial granularity with a spatial resolution or unit of, for example, 1 m$^2$, then the spatial change would have a large quantity, e.g., 1,000,000 cells. If a coarser spatial granularity with a spatial unit of 1 km$^2$ is applied, then the change will have a small quantity of 1 cell.

The relation between spatial granularity and spatial change is illustrated in Figure 1.1, where (a)-(b) have a coarser spatial granularity and (c)-(d) have a finer spatial granularity. Suppose that in Figure 1.1, cells in gray color represent forest coverage, then between (a)-(b) the forest coverage has decreased one cell, while between (c)-(d) the forest coverage has decreased four cells.

Secondly, the representation of change has temporal granularities. Continuing the logging example above, if the logging event is observed in a finer temporal granularity with a time interval of one hour, then it may be observed that the forest coverage is reduced by a rate of 100 m$^2$ per hour. If the logging event is observed by a time interval of one day, then the forest coverage could be reduced...
Figure 1.2: Temporal granularity is one of the three components of the granularity of change representation. (c)-(e) have a finer temporal granularity with a time interval of $\Delta t = t_2 - t_1 = t_3 - t_2$. (a)-(b) have a coarser temporal granularity with a time interval of $2\Delta t$.

by a rate of 1000$m^2$ per day.

Figure 1.2 illustrates different representations of changes with different temporal granularities. Suppose the observations have a fixed time interval $\Delta t$, i.e., $\Delta t = t_2 - t_1 = t_3 - t_2$, then Figure 1.2(c)-(e) have a temporal granularity with a time interval of $\Delta t$. The forest coverage (gray color) decreases two cells between Figure 1.2(c) and (d), and decreases other two cells between Figure 1.2(d) and (e). On the other hand, Figure 1.2(a)-(b) have a time interval of $2\Delta t$, and the forest coverage has decreased four cells.

Thirdly, different representations of changes with different thematic granularities are illustrated in Figure 1.3. Figure 1.3(a)-(b) have a coarser-grained measurement, and the natural phenomenon, e.g., temperature field, has been classified into two classes, e.g., cold (white) and warm (gray). Between Figure 1.3(a)-(b), three cold cells have changed to warm cells. If a finer-grained measurement is applied, the temperature field may be divided into three classes, e.g., cold (white), warm (gray), and hot (black). As shown in Figure 1.3(c)-(d), three cold cells have changed to warm cells, and three warm cells have changed to hot cells.

The discretization of continuous change by spatial, temporal, and thematic granularities of observations allows the distinction of two types of changes, i.e., abrupt spatial change and gradual spatial change, or simply *abrupt change* and *gradual change*. Both abrupt and gradual changes are discrete representations of the continuous change in the physical world.
Abrupt and gradual changes are distinguished by the quantity of change. Given certain spatial, temporal, and thematic granularities, a gradual change is the change of one and only one spatial cell, and an abrupt change is the change of more than one spatial cells. For example, the change in Figure 1.1(a)-(b) is a gradual change, since only one spatial cell is involved in the change. In Figure 1.1(c)-(d), four spatial cells have changed, therefore the change is an abrupt change.

The distinction between abrupt and gradual changes is also the distinction between previous work and the work in this thesis. As will be discussed in Chapters 2 and 3, previous work can only detect gradual change.

In contrast, this thesis proposes a new spatiotemporal data model in Chapter 4 to detect both abrupt and gradual changes, because continuous change is not necessarily gradual change [140]. An example is illustrated in Figure 1.3(a)-(b), where three spatial cells change between two consecutive time steps $t_1$ and $t_2$. Figure 1.3(a)-(b) can be a discrete representation of a continuously evolving temperature field.

This thesis focuses on the analysis of discrete spatial change with different granularity of observations, as shown in Figures 1.1, 1.2, and 1.3. There are two ontologies that are frequently used to analyze change, referred to SNAP and
The SNAP ontology has a snapshot-based view, where entities are organized as temporal sequences of snapshots. Snapshots are commonly used in geographical applications. Conventional geographical information systems (GIS), for instance, acquire spatiotemporal data by remote or in-situ observations of typically relative fine spatial, but coarse temporal granularity, and thus, represent spatial phenomena as static snapshots [101]. When comparing two snapshot sequences, changes may appear to be abrupt due to the low temporal frequency of observations. Moreover, interpolation of the snapshots is not always appropriate to reconstruct the underlying dynamic processes of reality. In principle, the snapshot approach is unable to trace events that occur between snapshots.

An event-based approach in the SPAN ontology is then becoming promising in theoretical studies, e.g., [139]. An event-based spatiotemporal data model, for example, is proposed by [100]. The model in [100] only records the timestamp when changes occur, therefore it can be understood as a chronicle-based model that is dual to the snapshot-based model [47]. In this case, changes or events can be explicitly stored in a compacted timeline, and additionally the redundancy of spatiotemporal data in the snapshot-based model can be reduced.

However, one of the issues that the event-based model needs to resolve is the spatial, temporal, and thematic granularities of observations. It is the spatial, temporal, and thematic granularities that constitute a change in spatiotemporal data representation. With certain spatial and temporal resolution of observation, events may not be recorded somewhere, sometime. If, in contrast, one may argue that continuous observation is possible with future technologies, then the problem of redundancy of observations will arise. Large amount of energy can be consumed to observe an environment where no change has occurred.

Therefore, there exist situations where data from applications, e.g., snapshots, cannot satisfy theoretic analysis, and theoretic models, e.g., the event-based models, are difficult to apply in real world applications. It is the emergence of new observation tools, e.g., wireless sensor networks (WSN), that bring the theory and application closer, since WSN can provide new options of spatial, temporal and thematic granularities of observations. A WSN is a network of computing devices that can collaborate via radio communication. A WSN is also a network of observation devices, since each node in the WSN is equipped with sensors that enable thematically fine-grained observation of the environments. A WSN is able to observe change in real-time, and hence, with almost any temporal granularity (but typically coarser spatial granularity).

In the context of geographic information science (GIS), the snapshot-based approach and the event-based approach have been used to model WSNs. The snapshot-based approach is commonly used in applications, e.g., [132], where
sensor nodes are tasked to periodically sense and store snapshots of an environment by setting the sensor network to a certain temporal sensing resolution. Although the snapshot approach is more practical, theoretical studies, e.g., [140], are more interested in the event-based model of WSNs due to its advantages of detecting salient changes or events. But, as discussed above, the event-based approach is based on the underlying assumption of continuous observation, which is not practical in real world scenarios. Moreover, the redundancy of continuous observations becomes more critical in the case of WSNs, since WSNs are battery-powered and energy resources are highly constrained for sensor nodes.

It is under these circumstances that this thesis is motivated. The goal of this thesis is to develop a spatiotemporal data model for WSNs that incorporates both snapshot-based and event-based approaches, such that the model can adapt to different spatial, temporal and thematic granularities of observations, can detect, store, and analyze changes or events, and can reduce the redundancy of spatiotemporal data.

1.2 Problem Statement

A large amount of existing work on WSNs in the GIS domain, e.g., [27, 70, 109], uses event-based approaches and focuses on qualitative information and qualitative reasoning of spatial objects. One of the focuses of qualitative reasoning is detecting topological events, such as splitting, merging, appearance, and disappearance. For example, in a bushfire emergency response scenario, the split of fire regions might be important for allocating firefighters. However, one of the important aspects that event-based approaches ignores is the quantitative information, for instance the perimeter and area of a spatial region. For example, in a fire event, it might also be important for firefighters to be aware that the area of a fire region has increased from $1 \text{ km}^2$ to $10 \text{ km}^2$.

The importance of qualitative approaches has been emphasized, but qualitative approaches should not be a substitute for quantitative approaches. Qualitative and quantitative reasoning should be complementary methods [36]. This research, with the objective of incorporating event-based and snapshot-based approaches, also aims to integrate quantitative information into qualitative reasoning. The research then faces the challenge that with different granularity of observations the proposed data model should enable qualitative and quantitative reasoning in both cases of abrupt and gradual changes. Hence, this thesis has the following research question:

In both cases of abrupt and gradual changes, how can wireless sensor networks efficiently support qualitative and quantitative reasoning at different spatial, temporal, and thematic granularities?
And the hypothesis is:

A spatiotemporal data model for wireless sensor networks can provide the capability of efficient qualitative and quantitative reasoning at different spatial, temporal, and thematic granularities in both cases of abrupt and gradual changes.

In the hypothesis, it is expected that the WSNs can be efficient for both qualitative and quantitative reasoning. The efficiency of WSNs will be discussed in the next section.

1.3 Approach

For geographic data representation, there is a distinction between field-based and object-based models [54]. In the field-based model, each spatial location is mapped to a value selected from a particular attribute domain. On the other hand, in the object-based model, a spatial object is an individual entity with its identity, spatial locations, and other attributes [20].

The spatiotemporal data model proposed in Chapter 4 is an object-based model. Accordingly, an object-oriented approach is used to implement the proposed model. For example, in Chapter 4, sensor nodes in a WSN are modeled as a set of point objects with point observations. Two key components of an object are attribute and behavior [35, 137]. Each point object would have some attribute data, such as its identity, coordinates, and observational data obtained by sensors. A particular behavior of an object may be activated by the change of its attribute data. For example, a reporting behavior of a point object \( p \) may be activated when its observed temperature increases from 19.9\(^\circ\)C to 20.1\(^\circ\)C, and crosses the designed threshold of 20\(^\circ\)C. To interact with other objects, the point object \( p \) might send a message to other point objects, such that other point objects can activate particular behaviors accordingly.

An appropriate data structure is essential for efficient interaction or communication among point objects. The primary structure in the proposed spatiotemporal data model is edge. An edge consists of two points, i.e., a start point and an end point. By sensor network point observations, geographic phenomena will be abstracted as areal objects that consist of points, and these points could be organized as sequence of edges, as will be discussed in Chapter 4. Thus, the edge structure is the connection between point objects, i.e., sensor nodes in a WSN, and areal objects, i.e., the discrete representation of geographic phenomena.

With some temporal granularity, a sequence of snapshots will be acquired by sensor networks. Each snapshot in the sequence will contain a set of areal objects, and each areal object would contain a set of edges. By the evolving of areal objects over time, such as expansion, contraction, split, and merge, edges could remain unchanged, or be inserted or deleted.
Figure 1.4 shows two examples. In the first example, an areal object is expanded from Figure 1.4(a) to (b), so that edges $hj$, $jk$, $kl$, and $la$ have been inserted and edges $hi$ and $ia$ have been deleted. And the other edges remain unchanged. In the second example, the merge of two areal objects from Figure 1.4(c) to (d) involves the insertion of edges $ab$, $bc$, $cd$, $gh$, and $hj$, and the deletion of edges $ji$, $ia$, $gm$ and $md$.

The insertion and deletion of edges have great importance to the algorithms proposed in Chapter 5. Thus, the time when an edge is inserted or deleted will be recorded in the sensor network. Note that the storage of spatiotemporal data will be distributed in each point objects in the sensor network. As an example, the point object $a$ in Figure 1.4(b) will store the timestamp for the inserted edge $la$ and deleted edge $ia$. More precise details will be discussed in Chapter 4. Due to the possibility of abrupt changes, a large number of edges can be inserted and deleted at the same snapshot. A single point object is then possibly not able to locally support qualitative or quantitative analysis based on its own datasets. In this case, aggregations of distributed datasets from multiple point objects become necessary for qualitative or quantitative analysis.

Based on the spatiotemporal data model in Chapter 4, a set of decentralized algorithms for sensor networks is proposed in Chapter 5 to support qualitative and quantitative analysis of dynamic areal objects. The decentralized algorithms are designed to improve the efficiency of interaction or communication among point objects. The efficiency of the algorithms will be evaluated at
1.4 Major Results

The major contribution of this thesis to geographical information science is a methodology for spatiotemporal data representation, storage, and analysis in wireless sensor networks. The major results of this thesis are:

1. The distinction and definition of two change representations, i.e., abrupt and gradual changes, with spatial, temporal, and thematic granularities.
2. A spatiotemporal data model for change representation in both cases of abrupt and gradual changes.
3. A set of decentralized algorithms for qualitative and quantitative reasoning in both cases of abrupt and gradual changes.
4. A new network structure, called multi-granularity sensor network, which is used to test the efficiency of the proposed decentralized algorithms with different spatial and temporal granularities.

These results can be applied for a sensor network deployment that is able to track dynamic environment phenomena, detect and report qualitative and quantitative changes, and store the change of environment phenomena in a decentralized database. Furthermore, this research provides a foundation for the future investigation on a general and practical model for spatiotemporal databases.

1.5 Organization of the thesis

Chapter 2 reviews the related work on the representation and detection of spatial change. Firstly, the representation of spatial changes in geospatial databases is discussed. The commonly-used network structures and algorithms for wireless sensor network are then introduced. Finally, the monitoring of environmental changes using wireless sensor networks is discussed.

In Chapter 3, the representations of continuants and spatial changes are firstly defined. The granularities of observations are then defined. Two types of change representations, i.e., abrupt and gradual changes, are distinguished in Section 3.4 based on the granularities of observations. Finally, Section 3.5 introduces topological and non-topological changes in both cases of abrupt and gradual changes.

In Chapter 4, a new spatiotemporal data model is proposed to represent spatial changes in a WSN. Boundary edges are the primary structure in the new data model. An important behavior of sensor nodes, i.e., traversals of
boundary edges, is also introduced. Section 4.3 has a further discussion about granularity. A new sensor network structure, called multi-granularity sensor network, is introduced in Section 4.4 for the further investigation of the relations among spatial change, spatial data, and the granularity of observations.

Chapter 5 presents a new decentralized change detection algorithm for the detection, storage, and analysis of spatial changes. Firstly, the change detection algorithms can be used to construct boundary edges and calculate inserted and deleted edges, as introduced in Section 5.1. Section 5.3 introduces different types of changes, and Section 5.4 discusses how different kinds of changes can be represented and distinguished by sequences of inserted and deleted edges. The details of the change detection algorithm is then described in Sections 5.5 and 5.6. Furthermore, a new approach to store spatiotemporal data in a decentralized network is also proposed, as can be seen in Section 5.3. Finally, the algorithms to construct multi-granularity sensor networks is presented in Section 5.7.

Chapter 6 tests the proposed change detection algorithm in Chapter 5 in a simulated environment. Both qualitative and quantitative tests have been conducted in Chapter 6, and the evaluation results shows the algorithm is efficient for both qualitative and quantitative analysis. The experiments also show the flexibility of the algorithm for supporting both gradual and abrupt changes. Chapter 7 complements Chapter 6 by providing more experiments, and the results of these experiments are then discussed.

In Chapter 8, the work in this thesis is summarized, and the major contributions of this thesis are presented. The chapter concludes with the directions for future work.
Chapter 2

Literature Review

2.1 Spatial Change

The spatial change investigated in this thesis is the change of spatial entities. A change occurs whenever spatial entities possess different spatial attributes at different times [45]. Some examples of spatial attributes are location, orientation, size, shape, and connectivity.

2.1.1 Ontology of Change

Grenon and Smith [57] classify entities in the spatiotemporal world into two categories: one is continuants, e.g., the Earth, which exist over time through different sorts of change, and the other is occurrents, usually referred to as processes or events, which occur in time and unfold through a period of time. Also, occurrents depend on continuants as their bearers. The study of this thesis is on the perspective of continuants that exist at a given time at a given level of granularity and undergo different types of changes over time.

Frank [42] suggests that an ontology for GIS should be built as a set of tiers of ontology. He proposes the 5-tier ontology: human-independent reality (Tier 0), observation of physical world (Tier 1), objects with properties (Tier 2), social reality (Tier 3), and subjective knowledge (Tier 4). The tiers of ontology relevant to this thesis are observation of physical world (Tier 1) and objects with properties (Tier 2). Spatiotemporal data acquired by observations are measurement values on some spatial, temporal, and thematic measurement scales. Chapter 3 will organize the observation data of the continuous reality into spatial objects with properties.

In [44], Galton studies the ways in which spatial regions change over time. He points already to the fact that physical processes of change (change in physi-
cal reality) exist “in a variety of degrees of continuity” (p. 1), which means they are all continuous if only studied in an appropriate level of temporal granularity. For example, the erosion of a coastline can be regarded as continuous. In contrast, he states there exist changes in social reality that happen discontinuous, e.g., the redrawing of a boundary due to the change of land parcel ownership. This thesis is restricted to (observing the) physical reality. Also, the notion of abrupt and gradual changes in this thesis is different from the above distinction of continuous and discrete changes. As already been discussed in Chapter 1, both abrupt and gradual changes are the discrete representations of continuous change in the physical reality.

2.1.2 Representation of Change

Current geographical information systems generally use two data representation structures, i.e., raster and vector. Raster data structure is an implementation of the field-based model, while vector representation is an implementation of the object-based model [2, 46]. Raster and vector data are separately stored in spatial databases due to the separation in the use of field-based and object-based models. Raster and vector data structures are used to represent the static space in geographical databases in the earliest geographic information systems [101].

Temporal capabilities in database systems can be found in the early transaction systems, such as banking systems, where detailed history is recorded such that individual changes through time can be preserved. These systems usually use relational models and can only store non-spatial data.

Sinton [119] defines geographic information with three attributes: (a) phenomena or objects being observed, (b) location of the phenomenon, and (c) the time of the observation. Geographical data in a GIS should consist of both spatial and temporal information. How spatiotemporal data should be represented in a database has been widely investigated, e.g., [82, 100, 126, 136].

Armstrong [5] identifies three components of change: the morphology (metrics), topology, and attributes of spatial objects. In Chapter 6, the decentralized algorithms proposed in Chapter 5 are evaluated by qualitative and quantitative analysis: the qualitative analysis focuses on topological changes of spatial objects, and the quantitative analysis is related to the morphological changes of spatial objects. The three components in [5] may or may not change over time, resulting in eight different scenarios of change [108]: no change, change in geometry, change in topology, change in attributes, change in geometry and topology, change in geometry and attributes, change in topology and attributes, and change in geometry, topology, and attributes.
Spatial objects may continuously change over time. Since conventional geographic databases capture change by snapshots—and this holds true even for moving object databases storing trajectories as sequences of spacetime points—originally it was sufficient to introduce temporal order [41] to reason about change.

Peuquet and Duan [100] propose an event-based spatiotemporal data model. They apply a time-based approach, and the top-level information is an event-list. Each element in the event-list contains a time value \( t_i \) and all changes that have been detected by observations at time \( t_i \). In other words, the changes related to times are explicitly stored. They state that their model can facilitate the analysis of change through time and the analysis of patterns of change through time. One of the limitations of their work is that the temporal granularity is not considered in their model.

Hornsby and Egenhofer [65] suggest a qualitative representation of change. Their notion of change is based on object identity. A unique identity for an object is commonly used in object-oriented approaches [76]. In [65], Hornsby and Egenhofer model geographical phenomena at a high level of abstraction as objects with identities, and model changes to geographical phenomena as changes to object identities. They define a set of change operations that either preserve or change identities. In this thesis, the notion of granularity is also realizing such an operator, deciding locally what is a sufficient change to trigger an update of an object in a central database. Only a few of their operations concern a single object (such as create or destruct), and the majority is about combining or splitting objects. Their work is relevant for applications of sensor networks tracking topological changes, e.g., [27, 110]. This thesis also applies their concept of identity in sensor networks for decentralized maintenance of spatial object identities.

In [123], Stell starts from a partially ordered time scale and dynamic objects with four fundamental operations on these objects (birth, death, merge and split), similar to [65]. He states that the data in a spatial database can be envisaged, at a conceptual level, as a set of objects with spatial and non-spatial attributes. Examples of objects could be a building, a road, or a town, etc. He then introduces granularity of sets of objects. Given a set of objects \( X \) with finer granularity, a new set of objects \( Y \) with coarser granularity can be formed from \( X \) by two operations, amalgamation and selection. For example, given \( X = \{a, b, c, d\} \), a new set \( Y = \{i, j\} \) can be constructed by the selection operation. In the set \( Y \), for example, \( i = a \) and \( j = c \), i.e., the elements \( i, j \in Y \) are selected from the elements in the set \( X \). This means his concept of granularity is based on sets. Hornsby and Egenhofer [67] model moving objects over multiple granularities. Their use of granularity relates to the cognitive aspects of selecting
the appropriate level of detail for a task. They state that shifting between
different spatial and temporal granularities is a necessary routine for many tasks.
Schmidtke and Woo [113] define spatial granularity based on a mereotopological
framework. In contrast, this thesis suggests two independent dimensions of
granularity, one being purely spatial and one being temporal, and they are kept
independent from objects but related to observations.

Another relevant concept comes from Cole and Hornsby: the *noteworthy*
event [19]. They define an event as noteworthy if it requires intervention, e.g., an
automated notification. The change detection algorithms proposed in Chapter 5
realize a corresponding concept. The algorithms identify occurrences of change
of a spatial and temporal granularity that needs to be reported to a central
database.

2.2 Wireless Sensor Networks

Wireless sensor networks (WSN) have a wide range of applications, e.g., en-
vironmental monitoring, precision agriculture, and target tracking. In these
applications, WSN are usually tasked to sense, gather, and transmit useful in-
fomation, such as temperature, in a monitored environment. The ability of the
WSN to sense extensive geospatial phenomena depends on a large number of
sensors in the network. The sensor nodes in a WSN are usually inexpensive,
battery-powered, and have limited processing and communication capabilities
[56, 92, 148].

2.2.1 Environmental Applications

Nittel *et al.* [96] define a specific type of wireless sensor network, called geosensor
network, in which the monitoring, aggregating, and analyzing of geospatial
information is fundamental. The sensor network used in this thesis can be con-
sidered as a geosensor network. An important task of geosensor networks is
to aggregate local spatiotemporal data obtained by individual sensor nodes to
produce globally meaningful information. In-network aggregations, discussed
in Section 2.2.2, are widely applied in geosensor networks for aggregating local
data in individual sensor nodes.

Duckham and Reitsma [29] apply a geosensor network to monitor a hydro-
logical system. They focus primarily on the comparison of the real-time data
sensed by a geosensor network and the predicted data generated by an environ-
mental model. Sensed data may be used to calibrate the environmental model.
On the other hand, feedback from environmental models to sensed data can
be used to detect and correct sensor errors. Ferentinos *et al.* [40] use wireless
sensor networks to monitor ocean currents. They state that major ocean currents are established using coastal radar, but the acquired information is coarse spatially and temporally. Thus they investigate the alternative deployment of inexpensive wireless sensors for fine-grained ocean monitoring.

Wark et al. [132] investigate the application of sensor networks in agricultural research and practice. They deploy a set of moisture sensors to monitor the state of pastures. The individual moisture readings from each sensor node are interpolated into a contour map of soil moisture. The real-time sensor readings are available on their website [114]. Mainwaring et al. [92] deploy a wireless sensor network at Great Duck Island, Maine, for habitat monitoring. Werner-Allen et al. [133] deploy a sensor network on an active volcano to monitor volcanic eruptions. A summary of sensor network applications in environmental monitoring can be found in [60].

2.2.2 In-network Aggregation

One of the characteristics of spatial data is spatial autocorrelation, i.e., data from locations near one another in space are likely to be similar [98]. Spatial autocorrelation introduces redundancy into spatial data. Thus, sensor nodes that are close to each other in an environment can acquire redundant spatial data. In-network aggregation approaches can aggregate similar data from neighboring nodes to reduce the volume of transmitted data, so that energy efficiency for WSNs can be achieved. In-network aggregation involves local communication among neighboring sensor nodes, and only relevant aggregation results are reported to the centralized system. This section discusses three commonly-used sensor network structures, i.e., tree, cluster, and planar graph, for in-network aggregation.

Tree

Tree is a commonly-used structure for in-network aggregation. A graph $G$ is a tree if $G$ is connected and $G$ has no cycles. An example of the tree structure is shown in Figure 2.1, where node $a$ is the root, and $e, g, h$, for example, are the leaf nodes. The depth of the tree is five.

An application of tree structure in sensor networks is the Tiny AGgregation (TAG) service introduced by Madden et al. [90]. In TAG, the sensor network is organized as a tree, where each sensor node has exactly one parent (except for the root node with no parents). The root of the tree is the node where the query is injected into the network and where the aggregation result is retrieved from the network. Usually, the root of the tree is called the sink of the sensor network. Each node senses some environmental data and the leaf nodes will
send the sensed data to their parents. Instead of forwarding all data, the parents of the leaf nodes will process the query in the network and aggregate the data from the leaf nodes using aggregation functions, such as \textit{count}, \textit{min}, \textit{max}, \textit{sum}, and \textit{average}. Only the aggregation results will be sent to the upper level. The efficiency of the TAG algorithm depends on the number of levels of the aggregation tree.

Another application of tree structure is the isobar mapping approach proposed by Hellerstein \textit{et al.} \cite{64}. Their approach can not only support the basic aggregation functions, such as \textit{min} and \textit{max}, but also support other functions, such as contour maps. Polygons are aggregated in the network to represent contour maps, therefore not only attribute data, e.g., temperature, but also \textit{x} and \textit{y} coordinates should be aggregated at each level of the tree structure.

One of the limitations of tree structure is that all the nodes in the network need to be involved in the aggregation process. In this case, Gao \textit{et al.} \cite{51} propose a sparse data aggregation approach, in which only a selected subset of nodes would participate in the aggregation. Based on the sensor measurements, a subset of sensor nodes in the network will be selected as hot nodes. These selected hot nodes will determine how the aggregation tree is formed. After the sparse aggregation tree is formed, data can be aggregated along the tree.

Soheili \textit{et al.} \cite{121} propose a SPIX tree structure. Each node in SPIX has a minimum bounded rectangle that covers itself and its descendants. They focus on the spatial aggregation query that includes an aggregation function, e.g., \textit{sum} and \textit{average}, and a user-defined region or window. The SPIX structure can calculate, for example, the average temperature sensed by the sensor nodes within the user-defined window. Energy consumption can be reduced in their approach, because the minimum bounded rectangles will be used to select only a subset of relevant nodes for the aggregation process.

One of the problems of the tree structure is the ignorance of spatial charac-
teristics, e.g., spatial autocorrelation. Aggregation by tree structure may not be able to reduce the redundancy of spatial data, because two similar data near in space may belong to two different branches of the tree. For example, in Figure 2.1, nodes c and f are near each other in space and observe the same areal object, but c and f are located in two different tree branches. To overcome this problem, Yoon and Shahabi [144] introduce an improved tree structure. In the structure, nodes that are close to each other in space and sense similar values (within a given threshold) will be formed as a cluster. Each cluster will only report one sensor value, and the cluster will remain unchanged if the sensor value stays within a given threshold over time. As in Chapter 4, the major network structure used in this thesis is planar graph, which is different from the tree structures discussed above.

Cluster

Cluster, as used in [144], is another widely-studied structure for in-network aggregation. A clustered sensor network may form a tree, as in [144]. A wide range of algorithms are proposed to group sensor nodes into clusters. The cluster algorithms can be classified into two categories [1], i.e., non-overlapping algorithms, e.g., [6, 25, 26, 63, 129, 145], and overlapping algorithms, e.g., [14, 95, 146].

The non-overlapping algorithms aim at producing a minimal number of disjoint clusters. LEACH, developed by Heinzelman et al. [63], selects cluster heads using a probability function, and builds one-hop non-overlapping clusters. LEACH has the assumption that each node can one-hop communicate with its cluster head, and cluster heads can directly communicate with the base station. In this case, the data from all the nodes within a cluster can be aggregated and processed locally in the cluster head, and thus the redundancy of locally correlated data can be reduced. Figure 2.2(a) illustrates an example of non-overlapping clusters, where nodes a and b are cluster heads. Nodes c, d, e and f belong to the cluster of node b, because they are all within one-hop communication range of b.

In Section 5.7, a new hierarchical sensor network structure will be built based on the concept of clusters, so that the hierarchical structure can aggregate data in localities and efficiently monitor phenomena with different spatial granularities. However, this thesis does not assume that cluster heads have superior capabilities. All the nodes, including cluster heads, are homogeneous, e.g., they have the same communication range.

As another example of non-overlapping algorithms, Bandyopadhyay and Coyle [6] propose a randomized algorithm EEHC to build n levels of cluster
Figure 2.2: Examples of cluster structure. (a) Non-overlapping clusters. (b) Overlapping clusters. In (a) and (b), the large dashed-line circles represent one-hop communication ranges and two-hop communication ranges respectively.

hierarchy. EEHC is a bottom-up algorithm, where level one cluster heads are selected with certain probability $p$, and the level two cluster heads are chosen among the level one cluster heads, and so on. The hierarchical multi-hop structure makes EEHC more suitable for large networks. All the sensors in their algorithm have a fixed and limited radio range. Messages between two sensors that are not within each other’s radio range will be forwarded by other sensors.

Based on their EEHC algorithm, Bandyopadhyay et al. [7] later introduce an approach to determine sensor density and temporal sampling rate in a clustered network. In their approach, the consumed communication energy can be minimized by eliminating message collisions when an appropriate spatial and temporal sampling rate is determined. Although their focus is on package collision avoidance, their work provides another motivation for granular spatial and temporal observations.

Some overlap among clusters can facilitate inter-cluster communication. The ACE algorithm proposed by Chan and Perrig [14] allows overlapping clusters. Each cluster in the ACE algorithm consists of a cluster head and several followers, and the followers must be in the communication radius of the cluster head. Thus the shape of a cluster is a circle with one communication radius. Two circles (clusters) may overlap, so that there may be several cluster heads within the range of a follower. However, in the algorithm, each node must choose only one cluster head, and belongs to exactly one cluster.

Youssef et al. [146] propose the MOCA algorithm to organize sensor nodes into overlapping multi-hop clusters. MOCA is currently the only overlapping multi-hop clustering algorithm. In the algorithm, the overlapping $k$-hop clusters should satisfy two conditions: (a) each node is either a cluster head or within $k$-hop from a cluster head, and (b) each cluster is overlapping by at least one other cluster. In the MOCA algorithm, the cluster heads are randomly selected, while the algorithms for constructing hierarchical sensor networks proposed in
Section 5.7 select cluster heads based on local network topology. An example of the two-hop overlapping clusters is shown in Figure 2.2(b), where a and b are cluster heads, and any nodes that are within two-hop of the cluster head belong to the cluster. The intersection of the two clusters a and b are \{d, e, h, i, j, m, n\}.

**Planar Graph**

Sensor networks are usually modeled as planar graphs by GIS communities, e.g., [27, 110, 111, 140]. For example, Worboys and Duckham [140] model a sensor network as a triangulation network. Sadeq and Duckham [110] model a sensor network as different planar graph structures, such as Delaunay triangulation, Gabriel graph, and relative neighborhood graph. Most of these works will be discussed in Section 2.3. In this thesis, a sensor network is also modeled as a planar graph, as will be introduced in Chapter 4. Figure 2.3 shows an example of planar graph structure. Note that a tree is a specific type of planar graph. The planar graph used in this thesis has cycles, and thus is not a tree.

![Figure 2.3: An example of planar graph. Different from a tree structure, there are no hierarchical relations among nodes in the graph.](image)

Planar graph structure can be used for spatial interpolations. Sharifzadeh and Shahabi [117, 118] point out that traditional aggregation operators, e.g., average and sum, are not reliable methods to study environmental phenomena. They argue that the non-uniform distribution of the sensor nodes in the environment needs to be considered. In this case, they introduce spatial aggregations and use spatial interpolation methods to answer spatial aggregations. They use node locations to partition the monitored field into Voronoi cells, and also organized the sensor nodes into Delaunay triangulation. Both Voronoi diagram and Delaunay triangulation are used to interpolate the attribute values of unknown locations. Their interpolation approaches use spatial statistics and focus on quantitative analysis of static spatial data at a given time. In contrast, the quantitative approaches in this thesis analyze dynamic spatiotemporal data, e.g., the changes of polygonal areas over time.
Summary

In this section, tree, cluster, and planar graph structures are introduced. A comparison between tree structure and planar graph structure will be further discussed in Section 4.4. And the limitations of both tree and planar graph structures for aggregating granular spatiotemporal data will be discussed. A new network structure, called multi-granularity sensor networks, is then introduced to overcome these limitations.

2.3 Environmental Monitoring Using Wireless Sensor Networks

Distributed approaches in WSNs for environmental monitoring applications can be classified into two categories, snapshot-based approaches and event-based approaches.

2.3.1 Snapshot-based Approach

In snapshot monitoring [64, 86, 149] WSNs are designed to constantly monitor environmental variables, e.g., temperature, humidity and light, of given geographic regions, and to regularly report snapshots of the monitored region to a designated central computer via the sink of the network.

In [86] Lian et al. propose a gradient boundary detection approach, where snapshots of a monitored region are transmitted to the sink at certain time intervals. A snapshot at a given time \( t \) is abstracted as a contour map, which consists of gradient boundaries. For example, in a temperature monitoring application the temperature distribution is subdivided into different levels of gradients, e.g., 10-15°C, 15-20°C, and 20-25°C. Then, the monitored region can be divided into a set of sub-regions based on the different levels of gradients. The sensor nodes that are located on the boundaries of the sub-regions will construct the gradient boundaries. Only the nodes on the gradient boundaries need to report to the sink, which constructs a contour map.

Different from Lian et al., Zhao et al. [149] use an algorithm to periodically transmit snapshots to the sink by in-network aggregation. In their algorithm an aggregation tree is constructed for the in-network aggregation process, where every sensor node has to be aggregated along the tree towards the sink. During the aggregation, neighboring nodes with similar values will be combined as polygons, called scans. Furthermore, two similar and adjacent polygons along the aggregation tree can be merged into a larger polygon. The similarity of the values associated with two polygons is defined by a tolerance parameter. Under
different aggregation tolerances, the sink can obtain scans of the whole network with the tradeoff of scan accuracy and energy efficiency. The aggregation tree structure is also used in isobar mapping [64].

An unique work for collecting snapshot data is proposed by Skraba et al. [120]. Their method is called a sweep, i.e., a wavefront that traverses the whole network and passes all nodes in the network exactly once. Preprocessing is required to find the gradients of the field, so that the gradients can guide the sweep process. The sweep method can only collect static snapshots. For a new snapshot, a wavefront has to cover all the nodes again for data collection.

Sarkar et al. [112] apply the sweep method to construct iso-contours in a continuous scalar field. The iso-contours encode spatial structures of a scalar field, e.g., boundaries of the ‘hot’ regions. Given the constructed iso-contours of a snapshot, users can acquired contours at given values or ranges, and can find the descending or ascending path between local minimum and local maximum. However, their approach is limited to static snapshot. Iso-contours that can adapt to dynamic scalar fields have not been addressed in their work.

2.3.2 Event-based Approach

Worboys and Duckham [140] regard events as the user’s high-level domain conceptualizations that could be inferred from the low-level detail of spatiotemporal information. Different applications might be interested in different types of events derived from the dynamic changes of a spatial phenomenon.

In the NED algorithm [72] Jin and Nittel define an event based on a simple threshold value. For example, in a temperature field, a temperature threshold of 200°C is used to define a fire event. NED focuses on the detection of event boundaries with noisy sensor readings. The NED algorithm is only able to detect event boundaries in a static snapshot. The algorithm can not detect events for evolving spatial phenomena over time.

Sadeq and Duckham [109] are detecting topological events among dynamic areal phenomena. The fundamental topological events include appearance, disappearance, merge and split. On the other hand, examples of non-topological events are expansion and contraction. Similarly to Lian et al. [86], this approach involves the detection of a region’s boundary, where boundary nodes are identified based on their qualitative relationship to immediate neighbors. For example, if a node $v$ has at least one neighboring node on the opposite side of a boundary, then the node $v$ is a boundary node. Although the evaluation shows their algorithm is efficient for detecting different topological events, their approach is limited by the assumption of gradual changes. A gradual change is defined as a change of a single sensor status over the entire network at a time $t$. 

21
Sadeq and Duckham [110] investigate different sensor network structures for detecting topological changes or events. Several commonly-used neighborhood structures, such as Delaunay triangulation, Gabriel graph, relative neighborhood graph, and greedy triangulation, have been tested. Their experiment results show that Delaunay triangulation and greedy triangulation provide the best qualitative results for event detection, because both of the two structures have sufficient neighbors to distinguish topological and non-topological events. Their approach also assumes gradual changes. Their classification of spatial changes is based on the theoretical study of Jiang and Worboys [71].

Jiang and Worboys [71] formally classify and define different types of topological events. They argue that it is useful for information systems to support queries of spatial events. To enable an event-based approach, an important task is to specify and distinguish different types of spatial events. They especially focus on the classification of topological events, i.e., the events related to the changes in topological structure of spatial objects. A tree structure is used to represent topological relationships between areal objects, and topological changes are then represented as tree morphism. They define five types of basic changes, i.e., appearance, disappearance, merge, split, and no change, based on the properties of the tree morphisms. They also define complex changes as a sequence of basic changes.

Based on the formal classification of topological changes in [71], Jiang and Worboys [70] develop a decentralized approach for WSNs to detect different types of topological changes. They consider the sensing reports as a sequence of snapshots of the areal objects. The difference or change between two consecutive snapshots is called a basic transition. Their decentralized approach is also based on the assumption of gradual change. But different from [110], they define a gradual change as the change of a piece of the areal object topologically equivalent to a disk. In this case, multiple sensors, which are located inside the disk, can change sensor status at the same time step. Therefore, local aggregation is required in their algorithm to aggregate local datasets for change detection.

Duckham et al. [27] monitor change of spatial fields by a geosensor network. They reconfigure flexibly the communication structures within a geosensor network for most efficiently capturing the boundaries of a changing spatial field. They model a geosensor network as a triangulation network. The triangulation can change over time in response to the movement of spatial phenomena. They use a qualitative approach to monitor dynamic spatial fields, and argue that qualitative approaches have three advantages. Firstly, in-network processing qualitative information can be more efficient than processing quantitative information, because qualitative properties form a smaller discrete domain.
ondly, qualitative information can be generated from quantitative phenomena, but the converse is not true. Thirdly, dynamic spatial fields are complex. Information systems should be easier to design and use by adopting a qualitative approach.

Worboys and Duckham [140] then further develop a computational model of spatial change using triangulation and combinatorial maps. In their model, spatial objects are represented as a set of triangles, and changes of spatial objects are represented as the insertion or deletion of triangles from the set. However, their approach is also limited to gradual changes, i.e., the change of a single triangle at each time step. They define several local transition rules based on the assumption of gradual changes. By applying these transition rules, different types of changes, such as expansion, contraction, hole formation, split, and merge, can be modeled and detected.

2.4 Conclusion

Section 2.1.2 introduces different representations of spatial changes. Different from the existing work in Section 2.1.2, this thesis defines the representation of spatial changes based on observations. Observations have spatial, temporal, and thematic granularities. The existence of granularity in spatiotemporal data allows the distinction of two representations of spatial change, i.e., abrupt and gradual changes. The event-based approaches in Section 2.3.2 can only detect gradual changes. This thesis provides a data model and associated algorithms to detect both abrupt and gradual changes.
Chapter 3

Spatial Change

The change investigated in this thesis is the spatial change of entities. Grenon and Smith [57] classify entities in the spatiotemporal world into two categories: one is continuants, and the other is occurrents, as introduced in Section 2.1.1. The study of this thesis is on the perspective of continuants that exist at a given time at a given level of granularity and undergo different types of changes over time. The representations of continuants and spatial changes are defined in Sections 3.1 and 3.2 respectively. The spatiotemporal data that represents change is acquired by observations, therefore the representation of change is subject to spatial, temporal, and thematic granularities of observations. The granularities of observations are formally defined in Section 3.3.

Two types of spatial changes, i.e., abrupt and gradual changes, are then distinguished in Section 3.4 based on granularities of observations. The distinction between abrupt and gradual change is also the distinction between previous work and the work in this thesis. Previous work discussed in Section 2.3.2 can only detect gradual change, while the work in this thesis can detect not only gradual but also abrupt change. More detail will be discussed in Section 3.4.

3.1 Representation of Continuants

This thesis focuses on the monitoring of a specific type of continuants, i.e., environmental phenomena, using sensor networks. An environmental phenomenon is modeled as a spatial scalar field in a 2D space \( R^2 \). A spatial scalar field represents the variation of some scalar property over the plane \( R^2 \) [47]. Formally, the spatial scalar field is a function \( f \), whose domain is the plane \( R^2 \) and codomain is a class of scalar values \( A \):

\[
f : R^2 \to A
\]  
(3.1)
Given a location $loc \in \mathbb{R}^2$, the function $f$ should return a value of a phenomenon $f(loc)$ at the specific location. It is assumed that at a given time each sensor node in the WSN is able to monitor some environmental phenomenon, such as temperature or moisture, and obtain this scalar value at the sensor location. The set of all the sensor nodes in a WSN is denoted as $V$. For each node $v \in V$, a location function $l$ can return the location of the node $v$ in the plane $\mathbb{R}^2$ at a particular time:

$$l : V \times T \rightarrow \mathbb{R}^2$$  \hspace{1cm} (3.2)

Many phenomena are continuously changing over time, so the spatial scalar field should be a *dynamic field*, and the network of sensors $V$ is able to measure the dynamic field and obtain different values $A$ over time $T$. Based on the functions (1) and (2), the value of a phenomenon sensed by a sensor node $v \in V$ at a time $t \in T$ is $f(l(v, t))$, which is defined as a sensed function $j$:

$$j : V \times T \rightarrow A$$  \hspace{1cm} (3.3)

The sensed function $j$ is the composition of $f$ and $l$, denoted as $j = l \circ f$. The scalar values of a phenomenon provide a quantitative representation of the environment, while the environment can also be represented in a qualitative perspective. Note that both quantitative as well as qualitative descriptions have a granularity by discretization, as will be further discussed in the following sections. Quantitative data can be used to derive qualitative information, as, e.g., in the classification of remotely sensed imagery [45, 54, 102]. For example, let us assume that the scalar values $a$ can be classified into a binary taxonomy $Y = \{y_1, y_2\}$. The elements in $Y$ can be, for example, $y_1 = \text{“low temperature”}$ and $y_2 = \text{“high temperature”}$. Thus, there is a function $k$ mapping from $A$ to $Y$:

$$k : A \rightarrow Y$$  \hspace{1cm} (3.4)

This thesis makes a further assumption that a specific set of threshold values, i.e., $\gamma_1, \ldots, \gamma_j$, is predefined by a domain expert to distinguish $y_i \in Y$. In the given example above, only one threshold value is required, e.g., $\gamma_1 = 20^\circ\text{C}$. If $a < \gamma_1$ then $k(a) = y_1$, and $k(a) = y_2$ otherwise. Given the Functions 3.3 and 3.4, there is a new function:

$$h : V \times T \rightarrow Y, \text{ where } h = j \circ k$$  \hspace{1cm} (3.5)
3.2 Representation of Changes

To track changes in the spatial scalar field, let us consider a finite region $X$, a subset of the 2D space $R^2$. Based on a taxonomy $Y$ the region $X$ can be modeled as a partition $S$, denoted as $S = \{s_1, s_2, ..., s_n\}$, such that a mapping $g'$ exists:

$$g' : S \rightarrow Y$$

For example, in Figure 3.1, a finite region $X$ has been subdivided into 16 cells, i.e., $S = \{s_1, s_2, ..., s_{16}\}$. Seven cells have a taxonomy type $y_1$, while the other nine cells have $y_2$, e.g., $g'(s_1) = y_1$ and $g'(s_3) = y_2$. Note that the partition of a region $X$ investigated in this thesis is not imposed by human beings, e.g., countries, states, or counties, but defined by the environmental phenomenon.

Furthermore, all $s_j$ with the same taxonomical type $y_i$ are grouped into a region collection $r_i$. Formally, a region collection is defined as $r_i = \{s_j : g'(s_j) = y_i, y_i \in Y\}$. A region collection $r_i$ does not need to be connected. In Figure 3.1, $r_1 = \{s_1, s_2, s_5, s_{11}, s_{12}, s_{15}, s_{16}\}$ and $r_2 = \{s_3, s_4, s_6, s_7, s_8, s_9, s_{10}, s_{13}, s_{14}\}$. The set of region collections $r_i$ is denoted as $R$, and the function $g'$ can be re-defined as a bijective function $g$ mapping from $R$ to $Y$, i.e., $g : R \rightarrow Y$, where $g(r_i) = y_i$. The inverse function of $g$ is $g^{-1} : Y \rightarrow R$, e.g., $g^{-1}(y_1) = r_1 = \{s_1, s_2, s_5, s_{11}, s_{12}, s_{15}, s_{16}\}$ as in Figure 3.1.

Since environmental phenomena are changing over an observation period $T$, the set of region collections $R$ will also change. In this thesis, the observation period $T$ is considered to be finite. Thus a new function $g_t^{-1}$ is defined as follows:

$$g_t^{-1} : Y \times T \rightarrow R$$

$R$ is the set of all region collections over time, i.e., $R = \bigcup_{t \in T} R(t)$. The function $g_t^{-1}$ can return a region collection for a taxonomical type $y_i \in Y$ at a particular
time \( t \in T \). Based on the function \( g^{-1}_t \) the regions that have changed over time from one taxonomical type to the other can be calculated. For example, given \( Y = \{y_1, y_2\} \), the regions that have changed from \( y_2 \) at \( t_i \) to \( y_1 \) at \( t_{i+1} \) are 
\[ g^{-1}_t(y_1, t_{i+1}) - g^{-1}_t(y_1, t_i). \]

Figure 3.2 illustrates an example about the change of region collections over time. Given \( Y = \{y_1, y_2\} \), there are \( g^{-1}_t(y_1, t_1) = \{s_1, s_2, s_5, s_{11}, s_{12}, s_{15}, s_{16}\} \) at \( t_1 \), and \( g^{-1}_t(y_1, t_2) = \{s_1, s_2, s_5, s_6, s_{10}, s_{11}, s_{12}, s_{15}, s_{16}\} \) at \( t_2 \), as in Figure 3.2(a) and (b) respectively. Thus, the regions that have changed from \( y_2 \) to \( y_1 \) are 
\[ g^{-1}_t(y_1, t_2) - g^{-1}_t(y_1, t_1) = \{s_6, s_{10}\}. \]

While phenomena in the physical world may undergo continuous change, sensor network observations have discretized and quantified the change spatially, temporally, and thematically. Therefore the change of region collections, e.g., \( g^{-1}_t(y_1, t_{i+1}) - g^{-1}_t(y_1, t_i) \), is dependent to spatial, temporal, and thematic granularities of observations. In the next section, granularities of observations are formally defined.

### 3.3 Granularity of Observation

Reitsma and Bittner [105] state that the observation of continuants at different level of granularity is critically important, since geographical information science is an empirical science. To approach granularity in the observation of continuants in sensor networks let us consider the following definitions:

**Definition 1** Temporal granularity \( \delta_t \) of an observation is the (average or regular) number of observations during an observation period, i.e., the frequency of observations.

**Definition 2** Spatial granularity \( \delta_x \) of an observation is the (average or regular) spatial density of sensor nodes in the area of the studied phenomenon.
Definition 3 Thematic granularity \( \delta_s \) of an observation is the (average or regular) sensitivity of the sensors.

These definitions allow us to distinguish between changes that occur in reality but are below the temporal, spatial or thematic thresholds of being noted by the WSN, changes in reality that are noted by the WSN but not significant enough to be reported to a central computer and therefore withheld by sensor nodes, and noted changes that are noteworthy in Cole and Hornsby’s [19] sense such that they have to be reported to a central computer.

If a WSN is not appropriately setup, observations can generate artifacts suggesting change falsely (error of commission), or can miss relevant change (error of omission). For what follows it is assumed that the sensor network has been designed appropriately. Examples of errors of omission and errors of commission can be found in the case study in Chapter 7.

To define abrupt and gradual changes in the next section, it is assumed that thematic granularity \( \delta_s \) of an observation allows the classification of the scalar values into a binary taxonomy \( Y = \{ y_1, y_2 \} \). A change from one taxonomical type to the other, e.g., from \( y_2 \) to \( y_1 \), is defined as a taxonomical change.

Based on the definitions of \( \delta_t \) and \( \delta_x \), the temporal and spatial granularities of sensor network observations can be calculated by the equations below:

\[
\delta_t = \frac{|\text{observations}|}{T} \quad (3.8)
\]

\[
\delta_x = \frac{|V|}{\text{area}(X)} \quad (3.9)
\]

where \(|\text{observations}|\) represents the number of observations during finite time \( T \), \(|V|\) represents the number of sensors deployed in the region \( X \), and \( \text{area}(X) \) represents the area of the region \( X \) in terms of, for example, square meters. Thus:

\[
T = \frac{|\text{observations}|}{\delta_t} \quad (3.10)
\]

\[
\text{area}(X) = \frac{|V|}{\delta_x} \quad (3.11)
\]

Based on Equations 3.10 and 3.11, the (average or regular) observation interval and the (average or regular) unit area can be calculated as follows:

\[
\Delta t = \frac{T}{|\text{observations}|} = \frac{|\text{observations}|}{\delta_t \times |\text{observations}|} = \frac{1}{\delta_t} \quad (3.12)
\]
The region $X$ has been subdivided into 16 cells, and there are 16 nodes in a sensor network. There is a one-to-one correspondence between nodes and cells. Sensor nodes, e.g., $a$, $b$, $c$, and $d$, are represented as small squares.

$$\Delta s = \frac{\text{area}(X)}{|V|} = \frac{|V|}{\delta_x \times |V|} = \frac{1}{\delta_x}$$

Temporally, it is assumed that the environmental phenomena are measured by the sensor nodes $V$ with a fixed time interval $\Delta t$, such that there is $t_{i+1} - t_i = \Delta t$ for any two consecutive time steps $t_{i+1}, t_i \in T$. At each time step, a snapshot of the phenomena will be acquired and recorded by the observation. Thus, with a certain temporal granularity $\delta_t$ of observations, environmental phenomena will be recorded as snapshot sequences. Figure 3.2(a)-(b), for example, is a sequence of two snapshots acquired by observations at time $t_1$ and $t_2$.

Spatially, the partition $S = \{s_1, s_2, ..., s_n\}$ defined in Section 3.2 should be determined by the spatial granularity. An example is shown in Figure 3.3, where there are 16 sensor nodes deployed in the region $X$, with spatial granularity $\delta_x = \frac{|V|}{\text{area}(X)} = \frac{16}{\text{area}(X)}$ as in Equation 3.9. Figure 3.3 also shows that the region $X$ is subdivided into 16 equal-size cells, i.e., $s_1$, $s_2$, ..., $s_{16}$, so that $\text{area}(s_1) = \text{area}(s_2) = ... = \text{area}(s_{16}) = \frac{\text{area}(X)}{16} = \frac{1}{\delta_x} = \Delta s$. Thus, in Figure 3.3, the partition of region $X$, i.e., $S = \{s_1, s_2, ..., s_{16}\}$, is derived from the spatial granularity of the sensor network $\delta_x = \frac{16}{\text{area}(X)}$. In other words, there is a one-to-one correspondence between sensor nodes and cells, as shown in Figure 3.3.

Note that the spatiotemporal data model introduced in the next chapter does not assume the regular grid distribution of sensor nodes. An example of irregular node distribution can be found in Figure 4.1. One of the approaches to derive partitions from point observations is Voronoi diagrams [53].
3.4 Abrupt and Gradual Changes

Based on the definitions of granularities of observations in the previous section, abrupt and gradual changes can then be defined:

**Definition 4** If $\exists t_i \in T$ such that $|g_{t_i}^{-1}(y_1, t_{i+1}) - g_{t_i}^{-1}(y_1, t_i)| = 1$ (given $Y = \{y_1, y_2\}$), then there is a gradual change from $y_2$ to $y_1$, and vice versa.

**Definition 5** If $\exists t_i \in T$ such that $|g_{t_i}^{-1}(y_1, t_{i+1}) - g_{t_i}^{-1}(y_1, t_i)| > 1$ (given $Y = \{y_1, y_2\}$), then there is an abrupt change from $y_2$ to $y_1$, and vice versa.

In the above definitions, $|g_{t_i}^{-1}(y_1, t_{i+1}) - g_{t_i}^{-1}(y_1, t_i)|$ represents the number of elements in the set $g_{t_i}^{-1}(y_1, t_{i+1}) - g_{t_i}^{-1}(y_1, t_i)$. Figure 3.4 shows examples of abrupt and gradual changes. In Figure 3.4(a)-(b), there is a gradual change, since $|g_{t_i}^{-1}(y_1, t_2) - g_{t_i}^{-1}(y_1, t_1)| = |\{s_6\}| = 1$, i.e., there is only one cell in the partition involved in taxonomical change. While, there is an abrupt change in Figure 3.4(c)-(d), because $|g_{t_i}^{-1}(y_1, t_2) - g_{t_i}^{-1}(y_1, t_1)| = |\{s_6, s_{10}\}| = 2 > 1$.

As already discussed in Section 2.3.2, all the previous work for WSNs can only detect gradual change. This thesis proposes a new spatiotemporal data model in Chapter 4 to detect both abrupt and gradual changes. There are three
major reasons to detect not only gradual change but also abrupt change, as follows.

Firstly, gradual change is a special case of continuity, but continuous change is not necessarily gradual change [140]. Figure 3.5 shows an example of continuous but not gradual change. Since $|g^{-1}(y_1, t_2) - g^{-1}(y_1, t_1)| = |\{s_3, s_7, s_{11}, s_{15}\}| = 4$, there is an abrupt change from Figure 3.5(a) to (b). Note that the continuous field has been discretized into a set of cells in Figure 3.5.

Secondly, there are always tradeoffs between granularity of observations and granularity of spatiotemporal data. If a finer granularity of observation is applied to an observation tool, e.g., a sensor network, more energy will be consumed for observation, communication, and data processing in the network, but the acquired data would have higher quality. On the other hand, coarser granularity of observation means less energy consumption and lower data quality.

With some temporal granularities of observations, changes may be detected and recorded as gradual or abrupt changes in a snapshot sequence. For example, in the second row of Figure 3.6, three gradual changes, i.e., the changes of cells $s_{10}$, $s_6$, $s_7$ respectively, have been detected in a sequence of four snapshots. While the first row of Figure 3.6 has a coarser temporal granularity of observation, and an abrupt change of three cells have been detected. Figure 3.6 demonstrates different granularities of change: the change with a coarser granularity in the first row, i.e., abrupt change, is composed by a set of changes with a finer granularity in the second row, i.e., gradual changes.

With temporal granularity of observation, several gradual changes may be aggregated into an abrupt change. Thus, both abrupt and gradual changes can exist in snapshot sequences. With granularity of observations, an approach to detect both abrupt and gradual changes is essential.
Thirdly, the data model proposed in the next chapter aims to be a practical model not only for sensor network applications but also for geographic databases. The global snapshot views of geographic data in low temporal resolution let all change appear to be abrupt. By supporting both gradual and abrupt changes, the data model provides a foundation for potential future work, such as change detection in realistic datasets, as will be discussed in Chapter 8.

3.5 Topological and Non-topological Changes

Six topological changes and two non-topological changes of region collections will be primarily investigated in this thesis. The topological changes of region collections can be classified into six fundamental types, i.e., appearance, disappearance, merge, split, self-merge and partial-split [71, 110]. The two non-topological changes are expansion and contraction.

Each of these eight types of changes can be either abrupt or gradual. For example, in Figure 3.8(a)-(b), there is an expansion of a region collection. Since

![Diagram of topological changes and non-topological changes](image)

Figure 3.7: Six topological changes and two non-topological changes.
Figure 3.8: (a)-(b) The expansion is gradual. (c)-(d) The merge is abrupt.

\[ |g^{-1}_{t_1}(y_1, t_2) - g^{-1}_{t_1}(y_1, t_1)| = |\{s_8\}| = 1, \] the expansion is gradual. In Figure 3.8(c), the region collection \( g^{-1}_{t_1}(y_1, t_1) = \{s_5, s_7, s_9, s_{11}\} \) has two disconnected components, i.e., \( \{s_5, s_9\} \) and \( \{s_7, s_{11}\} \), and in Figure 3.8(d), the two components have been merged into one connected component. Since \( |g^{-1}_{t_2}(y_1, t_2) - g^{-1}_{t_1}(y_1, t_1)| = |\{s_6, s_{10}\}| = 2, \) the merge is abrupt.

This thesis focuses on detection of topological and non-topological changes of region collections in both cases of abrupt and gradual changes. A connected component of a region collection is defined as an *areal object*. Thus, a region collection can consist of multiple disconnected areal objects.
Chapter 4

Spatiotemporal Data Model

In this chapter, a spatiotemporal data model is proposed to represent both abrupt and gradual changes in a sensor network. Based on the data model in this chapter, the next chapter will develop an algorithm for sensor networks to detect these two types of changes in a decentralized manner.

4.1 Sensor Network

As introduced in Section 1, a wireless sensor network is a network of untethered, battery-powered, and sensor-enabled computing devices. In this thesis, it is assumed that WSNs are static, and each sensor node knows its own geographic location. It is also assumed that each node is able to communicate information with its immediate neighbors using radio communication.

4.1.1 Sensor Network Model

A WSN consists of a set of sensor nodes that can make point observations of the environment. In this thesis, sensor nodes in a WSN are modeled as a set of point objects. Each point object has its unique identity and coordinates. Also each point object will contain observational data that is acquired by sensors.

A graph is used to define the neighborhood structure of a set of point objects. A WSN is modeled as a directed planar graph $G = (V, E)$. In the graph $G$, $V$ is a set of nodes and $E$ is a set of directed edges, e.g., $(v, v')$, which represent the communication links between the nodes $v \in V$ and $v' \in V$. $E$ is assumed to be symmetric, i.e., if there exist $v, v' \in V$ such that $(v, v') \in E$, then $(v', v) \in E$. The edge $(v, v') \in E$ is defined as an \textit{out} edge of $v$ to $v'$, and $(v', v) \in E$ is defined as an \textit{in} edge of $v$ from $v'$. Note that the direction of a directed edge will be illustrated in a figure when it is relevant, otherwise the representation
of $G$ can be simplified as in Figure 4.1(a).

The algorithms introduced in Chapter 5 are decentralized algorithms in the sense that each point object only stores information about itself and its immediate neighbors [84]. Each point object has its own datasets. Global knowledge, e.g., the information about the whole network $G$, is unavailable in point objects. The datasets stored in point objects are discussed below.

The set of immediate neighbors of a node $v \in V$ is denoted as $\text{neighbor}(v) = \{v' : (v, v') \in E\}$. For example, in Figure 4.1(b), $\text{neighbor}(v) = \{a, b, c, d, e\}$. The set $\text{neighbor}(v)$ is sorted into clockwise order, as shown in Figure 4.1(b).

The set of directed edges of a node $v$ is denoted as $\text{edge}(v)$. Since $E$ is symmetric, the set $\text{edge}(v)$ can be derived from $\text{neighbor}(v)$, i.e., $\text{edge}(v) = \{(v, v') : v' \in \text{neighbor}(v)\} \cup \{(v', v) : v' \in \text{neighbor}(v)\}$. In Figure 4.1(c), for example, the node $v$ has five neighbors, so that $v$ has ten directed edges $\text{edge}(v) = \{(v, a), (a, v), (v, b), (b, v), (v, c), (c, v), (v, d), (d, v), (v, e), (e, v)\}$. The set $\text{edge}(v)$ is also sorted in clockwise order, and for a same neighboring node $v'$, the in edge $(v', v)$ is always the next edge of the out edge $(v, v')$, as in Figure 4.1(c).

A node $v$ and its neighbors are organized into directed cycles, or simply cycles. A cycle is denoted as $(v_0, ..., v_{i-1}, v_i, ..., v_n)$, where all nodes are distinct except $v_0 = v_n$, and there is an edge for any two consecutive nodes in the cycle, for example, $(v_0, v_1), (v_1, v_2), (v_{n-1}, v_n) \in E$. It is defined that $v_i \in (v_0, ..., v_{i-1}, v_i, ..., v_n)$ and $(v_{i-1}, v_i) \in (v_0, ..., v_{i-1}, v_i, ..., v_n)$. An example of cycles is shown in Figure 4.1(c), where $(v, e, d, v)$ is a cycle with three nodes. The set of cycles of a node $v$ is denoted as $\text{cycle}(v)$. For example, in Figure 4.1(c), the set $\text{cycle}(v) = \{(v, a, c, v), (v, b, a, v), (v, c, b, v), (v, d, c, v), (v, e, d, v)\}$. Given a cycle, e.g., $(v, b, a, v)$, it is also defined that $(v, b, a, v) = (b, a, v, b) = (a, v, b, a)$.

Since $\text{edge}(v)$ is in clockwise order, all the cycles $\text{cycle}(v)$ of a node $v$ are coun-
terclockwise, as illustrated in Figure 4.1(c). The cycle structure is important for the traversal of the graph $G$.

### 4.1.2 Traversals

The algorithms proposed in the next chapter are distributed algorithms since they run in parallel on all point objects at the same time. Individual datasets stored in point objects may not be sufficient to support qualitative and quantitative analysis of dynamic areal objects. In-network processing and aggregation of spatiotemporal data among point objects thus become necessary.

The sensor network model uses the traversal of the directed planar graph $G$ for in-network data aggregation. The cycle dataset in each point object $v \in V$ is important for graph traversal. Firstly, the directed edges in $E$ are classified into two categories: cycle edges and non-cycle edges. A cycle edge is a directed edge that belongs to at least one cycle, and similarly a non-cycle edge is not in any cycles. If the whole set $E$ is considered, then almost all of the directed edges are cycle edges, except the edges at the boundary of the sensor network. But usually only a subset of $E$ is considered. For example, in Figure 4.2(a), suppose the directed edges in the areal object $A$ have been activated, and all the other edges in $E$ are inactive. Among the active edges in Figure 4.2(a), the edges $(v, e)$, $(e, d)$, and $(d, v)$ form a cycle $(v, e, d, v)$, and thus these three edges are cycle edges. The other five edges are non-cycle edges since they do not belong to any cycles in the areal object $A$.

![Figure 4.2: Traversals. (a) The traversal of non-cycle edges based on the clockwise rule. (b) A region with a hole can be detected by the traversal orientations.](image)

Only non-cycle edges will be used for traversals, so that the cycle edges are set to inactive, illustrated as dashed-line arrows in Figure 4.2(a). Suppose the traversal is started at the node $b$, then the route will be a closed path followed the clockwise rule: $b \rightarrow v \rightarrow d \rightarrow e \rightarrow v \rightarrow b$. Formally, the path of a set of connected non-cycle edges is a Eulerian trail, since the number of non-cycle
edges in each node is always even. An important property can be deduced from a closed traversable trail, i.e., the traversal orientation.

4.1.3 Traversal Orientation

Two types of traversal orientations are defined: clockwise and counterclockwise. Beyer [9] defines that the area of a polygon is positive if the vertexes of the polygon are arranged in a counterclockwise order, and negative if they are in clockwise order. The area of a polygon can be calculated by the function \[ \text{Area} = \frac{1}{2} \sum_{i=1}^{n} (x_i y_{i+1} - x_{i+1} y_i). \] Based on the above definition, the result of an area calculation can be used to determine the ordering of the vertices of a polygon, and thus the orientation of a traversal.

Importantly, a traversal of a set of non-cycle edges is directed. For example, in Figure 4.2(b), there is a region \( A \) with a hole. A region with a hole is usually represented by two polygons: one external polygon for the region and one internal polygon for the hole. Correspondingly, the sensor network would have one external and one internal traversal. As in Figure 4.2(b), suppose both traversals start at node \( a \), then the external traversal is \( a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow a \), and the internal traversal is \( a \rightarrow e \rightarrow d \rightarrow c \rightarrow b \rightarrow a \). There is a property that external traversals are clockwise and internal traversals are counterclockwise. With this property, the detection of a region with a hole is straightforward.

4.2 Change Representation in Sensor Network

In Section 3.3, it is assumed that thematic granularity \( \delta_s \) of an observation allows the classification of the scalar values into a binary taxonomy \( Y = \{y_1, y_2\} \). This thesis focuses on the change of areal objects, so it is further defined that \( y_1 \) represents areal objects of interest. Thus, the equation 3.5 can be simplified as \( h: V \times T \rightarrow \{y_1, y_2\} \), which represents that the sensed value at an observation point at a particular time can be classified into two classes \( y_1 \) or \( y_2 \). If a sensor node \( v \) has \( h(v, t) = y_1 \), then the node is located in the areal object at time \( t \).

4.2.1 Boundary Edges

Based on the above definition, the representation of areal objects in a sensor network can be defined. Areal objects are represented as sequence of directed edges in the network, and there are four definitions as follows:

**Definition 6** A directed edge \( (v, v') \in E \) is an object edge, if both \( v \) and \( v' \) are located in an areal object.
Definition 7 A cycle $(v_0, ..., v_{i-1}, v_i, ..., v_n)$ is an object cycle, if all the vertexes of the cycle, i.e., $v_0, ..., v_{i-1}, v_i, ..., v_n$, are located in an areal object.

Definition 8 An object edge $(v, v') \in E$ is a non-boundary edge, if the edge belongs to an object cycle.

Definition 9 An object edge $(v, v') \in E$ is a boundary edge, if the edge does not belong to any object cycles.

An example is shown in Figure 4.3(a), in which the nodes $v, b, c$ and $e$ are located in the areal objects. In this case, the edges $(v, b)$, $(b, v)$, $(v, c)$, $(c, v)$, $(v, e)$, $(e, v)$, $(b, c)$, $(c, b)$ are object edges. The cycle $(v, c, b, v)$ is an object cycle, since the nodes $v, c, b$ are all located in the areal object. Thus, there are three non-boundary edges, i.e., $(v, c)$, $(c, b)$, $(b, v)$ and five boundary edges, i.e., $(v, b)$, $(b, c)$, $(c, v)$, $(v, e)$, $(e, v)$. Note that boundary edge and non-boundary edge are specific definitions of cycle edges and non-cycle edges in Section 4.1.2. Boundary edges of areal objects are important for the algorithms in Chapter 5. Usually only boundary edges will be illustrated in the figure, as illustrated in Figure 4.3(b).

Figure 4.3: An areal object is represented as a sequence of directed edges in a sensor network. Directed edges within an areal object are classified into boundary edges (solid-line arrows) and non-boundary edges (dashed-line arrows), as illustrated in (a). Usually only boundary edges will be illustrated in the figure, as in (b).

As already discussed in Section 4.1.1, the algorithms in Chapter 5 are decentralized algorithms, therefore the boundary edges are decentrally stored in each relevant sensor node. Each node $v$ requires a data set, denoted as boundaryEdge$(v, t)$, to store its boundary edges. Following the example above, the dataset at sensor nodes $v$ and $b$, for example, would be boundaryEdge$(v, t) = \{(v, b), (c, v), (v, e), (e, v)\}$ and boundaryEdge$(b, t) = \{(v, b), (b, c)\}$.
4.2.2 Insertion and Deletion of Edges

The previous section has not discussed the change of areal objects over time. Since the proposed sensor network model allows granularities of observation, especially temporal granularity, the change of areal objects will be captured by a sensor network as sequences of snapshots. In each snapshot, sequences of boundary edges are used to represent areal objects. Since areal objects are evolving over time, the sequences of boundary edges would dynamically change at different snapshots, as illustrated in Figure 4.4(a)-(d). The difference of boundary edges between two consecutive snapshots is represented by the insertion and deletion of boundary edges. If a boundary edge appears between \( t_{i-1} \) and \( t_i \), then the boundary edge will be inserted into relevant datasets with a timestamp of \( t_i \), and vice versa. Each sensor node may be involved in the insertion and deletion of boundary edges in its own datasets.

To represent the change of areal objects in a sensor network, two definitions are required:

**Definition 10** A boundary edge \((v, v') \in E\) is an inserted boundary edge, or simply an inserted edge, at time \( t_i \) if the edge is not a boundary edge at previous time \( t_{i-1} \), but become a boundary edge at time \( t_i \).

**Definition 11** A boundary edge \((v, v') \in E\) is an deleted boundary edge, or simply a deleted edge, at time \( t_i \) if the edge is a boundary edge at previous time \( t_{i-1} \), but is not a boundary edge at time \( t_i \).

Each inserted and deleted edge will be given a timestamp in the datasets of each relevant sensor node. Two datasets are required in each sensor node, i.e., \( \text{insertedEdge}(v, t) \) and \( \text{deletedEdge}(v, t) \), to store the inserted and deleted edges over time. As an example in Figure 4.4, a sensor network detects the change of an areal object from \( t_0 \) to \( t_3 \). An areal object appears between \( t_0 \) and \( t_1 \), and the object expands between \( t_1 \) and \( t_2 \) and between \( t_2 \) and \( t_3 \).

In Figure 4.4(a), there is no areal object at \( t_0 \). In Figure 4.4(b), there are four inserted boundary edges due to the appearance of an areal object. These inserted edges will be recorded at different datasets, i.e., \( \text{insertedEdge}(v, t_1) = \{(v, b), (b, v), (v, c), (c, e)\} \) at \( v \), \( \text{insertedEdge}(b, t_1) = \{(b, v), (v, b)\} \) at \( b \), and \( \text{insertedEdge}(e, t_1) = \{(e, v), (v, e)\} \) at \( e \). At \( t_2 \), three nodes \( v, b, \) and \( c \) need to record the insertion and deletion of boundary edges. There are \( \text{insertedEdge}(v, t_2) = \{(c, v)\} \) and \( \text{deletedEdge}(v, t_2) = \{(b, v)\} \) for the node \( v \), \( \text{insertedEdge}(b, t_2) = \{(b, c)\} \) and \( \text{deletedEdge}(b, t_2) = \{(b, v)\} \) for \( b \), and \( \text{insertedEdge}(c, t_2) = \{(b, c), (c, v)\} \) for \( c \). Similarly, sensor nodes can record the changes of boundary edges at \( t_3 \).

As a summary, the recorded insertion and deletion of edges at a node, e.g., \( e \), would be:
The boundary edges at a sensor node at a particular time can be updated based on the inserted and deleted edges. And usually, only the boundary edges at current time step $t_i$, i.e., $boundaryEdge(v, t_i)$ will be maintained in the database. Boundary edges at other time steps can be calculated correspondingly. For example at $t_0$, $boundaryEdge(e, t_0) = \emptyset$. At $t_1$, the boundary edges of the node $e$ can be updated using set operations:

$$
boundaryEdge(e, t_1) =
boundaryEdge(e, t_0) \cup insertedEdge(e, t_1) - deletedEdge(e, t_1) =
\emptyset \cup \{(e, v), (v, e)\} - \emptyset = \{(e, v), (v, e)\}.
$$

The above examples demonstrate that each sensor node only possesses a subset of spatiotemporal data about the change of areal objects. To generate useful information from these decentralized spatiotemporal data, sensor nodes need to communicate with each other for data integration.

### 4.2.3 Abrupt and Gradual Changes in WSNs

Given $h : V \times T \rightarrow \{y_1, y_2\}$, if $h(v, t_{i-1}) \neq h(v, t_i)$ for a node $v \in V$, then the node $v$ detects a taxonomical change at the current time step $t_i \in T$. The class of a node $v$ may change from $y_2$ at $t_{i-1}$ to $y_1$ at $t_i$, i.e., the node $v$ is not located...
Figure 4.5: Gradual and abrupt changes in WSNs. Sensor nodes that are located in the areal objects are represented as solid squares. (a)-(b) is a gradual change. (c)-(d) is an abrupt change.

in areal objects at $t_{i-1}$, but it is located in an areal object at $t_i$, and vice versa.

If a node $v \in V$ detects a taxonomical change at time step $t_i \in T$, then the node $v$ is defined as an active node at $t_i$. The set $\text{active}(t)$ is used to represent all active nodes at the time step $t$. Obviously, $\text{active}(t) \subseteq V$. For example, in Figure 4.5(a)-(b), an areal object has expanded between $t_1$ and $t_2$, and the node $e$ has changed sensor readings from $y_2$ to $y_1$. Thus, node $e$ is an active node at $t_2$, i.e., $\text{active}(t_2) = \{e\}$.

Given the concept of active nodes, abrupt and gradual changes in WSNs can then be defined as follows:

**Definition 12** Between two consecutive time steps $t_i$ and $t_{i-1}$, a change of areal objects is a gradual change if there is one and only one active node at time step $t_i$.

**Definition 13** Between two consecutive time steps $t_i$ and $t_{i-1}$, a change of areal object is an abrupt change if there are more than one active nodes at time step $t_i$.

Gradual changes only allow one active node per time step. The decentralized algorithms proposed in Chapter 5 can be applied not only for a single active node, but also for a set of active nodes at each time step. Thus, the proposed algorithms are suitable for both gradual and abrupt changes. An example of
abrupt and gradual changes in WSNs is shown in Figure 4.5, where (a)-(b) is a gradual change with $active(t_2) = \{e\}$ and (c)-(d) is an abrupt change with $active(t_2) = \{d, e\}$.

The abrupt and gradual changes are related to the granularity of sensor network observations. For example, if a sensor network has a coarser spatial resolution, e.g., lower node density, only one sensor node may detect change of an areal object. If the sensor network has a higher node density, more than one node may be able to detect the same change. Similarly, a sequence of gradual changes may be detected by a sensor network with finer temporal sensing resolution, while the same sensor network with coarser temporal resolution may only detect an abrupt change. In the next section, the relation between change and granularity of observation will be discussed.

4.3 Change and Granularity

There is a relationship between the granularity of observations, granularity of spatiotemporal data, and granularity of spatial changes, as illustrated in Figure 4.6. Some examples have been provided to explain these relations in Chapters 1 and 3.

![Diagram](image)

Figure 4.6: There are relations between granularity of observation, granularity of spatiotemporal data, and granularity of spatial changes.

The relations between granularity of spatial changes and granularity of observations will be investigated in the algorithms in Chapter 5, which includes two parts. The first part in Sections 5.1 - 5.6 primarily investigates granular spatial changes with different temporal granularity of observations. The second part in Section 5.7 focus on the building of sensor network structure with different spatial granularity of observations.

With different temporal granularities of observations, spatiotemporal data acquired by sensor network can be represented as sequences of snapshots. Figure 4.7(a) and (b) shows two different snapshot sequences with two different tem-
Figure 4.7: (a) and (b) have different temporal granularities of observations, as will be investigated in Sections 5.1 - 5.6.

Figure 4.8: (a) and (b) have different spatial granularities of observations, as will be investigated in Section 5.7.

Temporal sensing resolutions. The first part of Chapter 5 will focus on the analysis of snapshot sequences with certain temporal granularity of observation.

The spatial granularity of observations is the density of sensor nodes in the sensor network deployed area, as defined in Section 3.3. In the second part of Chapter 5, the construction of planar graphs with different node densities will be investigated. Figure 4.8 shows an example of different spatial granularities of observations.

4.4 Multi-Granularity Sensor Networks

To investigate the relationship among spatial changes, spatiotemporal data, and observations, a new hierarchical sensor network structure, called a multi-granularity sensor network, is introduced. The new network structure will enable different spatial granularities of observations, and achieve communication efficiency. The new network structure is based on two commonly-used structures, i.e., the tree and planar graph.
Before the discussion of tree and planar graph structures, the conventional centralized structure for collection and interpretation of sensed data is firstly introduced. In the centralized approach the sensed data of a WSN need to be periodically transmitted to a sink in the network, and the sink could then communicate with a central computer, e.g., [92]. The frequent transmission of observation data to the sink will dramatically consume the constrained energy in sensor nodes, since wireless communication intensively demands energy in the sensors. Thus, the centralized sensed and transmitted approach is not energy efficient. Also, transmitting the large amount of spatio-temporal data produced by WSNs to a sink can result in significant information redundancy and communication overload.

The topology of the centralized sensor network structure is shown in Figure 4.9. The topology among sensor nodes and the sink can be represented as a tree structure, where the sink is the root and the sensor nodes are the leaves. As in Figure 4.9, the communication between a sensor node, e.g., b, and the sink a is hierarchical. An edge between a leaf node and the root node represents that the leaf node can communicate with the root node via, e.g., multi-hop communications.

![Figure 4.9](image.png)

Figure 4.9: A centralized structure. There are hierarchical communications between sensor nodes, e.g., b, and the sink a.

### 4.4.1 Tree and Planar Graph

In a tree structure as in Figure 4.10 (a) and (b), sensor nodes are organized as an aggregation tree, e.g., [90]. The root of the tree is the sink, and the precedence relationship between two nodes can be defined by, for instance, the geographic locations of the sensor nodes. Parent and child will be used to describe the precedence relationship between two nodes. For example, in Figure 4.10, a sensor network deployment region has been subdivided into a set of regular square areas. For each square area, one sensor node is selected as a parent node, and the other nodes in the area are the children of the parent. As shown in Figure 4.10, node a is selected as a parent node, and node b is the child of node a. Sensor nodes in a tree have hierarchical communications from the leaves
The communication in a tree structure is hierarchical. Thus, two nodes that are geographically close to each other may not be able to communicate, because they belong to different branches of a tree. An example is shown in Figure 4.11(a), where a sensor network is organized as a tree and monitors some areal objects. Four nodes $a$, $f$, $g$, and $h$ have detected an areal object, and they should report their sensor reading by in-network aggregation. Although the four nodes are close to each other, they will be separated into two groups, i.e., \{a\} and \{f, g, h\} for tree aggregation. As discussed in Section 2.2.2, the tree structure ignores spatial autocorrelation, and thus it will increase the redundancy of in-network aggregations.

Another example of environmental monitoring using tree structures is shown in Figure 4.11(b). One of the limitations of tree structure is that all the nodes located inside the areal object needs to be involved in the aggregation process. If there is a larger areal object compared to Figure 4.11(b), then hundreds of sensor nodes may be involved in aggregation, and the efficiency of tree structures will be significantly reduced.

In this circumstance, the planar graph structure is introduced for in-network aggregations. An example of planar graph can be found in Figure 4.1(a). Compared to tree structures, planar graph structures have the capability of peer-to-peer communications. Details of the planar graph have been introduced in Section 4.1.1. Figure 4.12 provide two examples of environmental monitoring using planar graph structures. Planar graph structures should be more efficient to monitor static spatial objects, because usually only the nodes on the boundary of the spatial object are involved in aggregation, as illustrated in Figure 4.12 (a) and (b).

However, the planar graph structures show limitations for monitoring dy-
Dynamic spatial objects. For example, an areal object with a circular shape appears and starts to expand in a region deployed with a sensor network, and the sensor network is tasked to monitor the size of the areal object. In the planar graph structure, the size of the circle can be calculated by aggregating the data along the boundary of the circle. However, if the circle expands continuously, the size of the circle need to be re-calculated by numerous in-network aggregations along the boundary. Although the planar graph structure can obtain the size of the circle by peer-to-peer communication, the frequent boundary traversals can become costly especially when the size of the circle is large. On the other hand, the tree structure can avoid frequent long distance traversals, since the parent nodes can aggregate change reports from their children, and only report to the upper level when the number of reports exceed a designed threshold.

As a summary, planar graph structures have the capability of peer-to-peer communications, and there are hierarchical communications in tree structures. The objective of the new network structure is to combine tree and planar graph structure, so that the new structure has both hierarchical and peer-to-peer communication capabilities. Different from previous work discussed in Section 2.2.2, the new network structure is designed using object-oriented approaches, as will be introduced in the next section.
To achieve the objective of incorporating both hierarchical and peer-to-peer communications, the new network structure is organized as multiple levels of planar graphs. Figure 4.13 shows a general idea about the construction of the new network structure. Given a planar graph as the first level of the new structure as in Figure 4.13(a), a subset of nodes is selected from the first level to build the second level in Figure 4.13(b). The second level is also constructed as a planar graph. Similarly, a subset of nodes from the second level can be selected to build the third level planar graph, and so on. The new network structure will include all these different levels of planar graphs.

![Figure 4.13: The construction of a new network structure. A subset of nodes in the first level (a) is selected to build the second level (b).](image)

In Sections 4.1-4.2, a new spatiotemporal data model has already been proposed for a one-level flat planar graph. Based on the object-oriented approach, the planar graph at each level of the new network structure can inherit the attributes and behaviors of the already developed planar graph. For example, if there is a three-level hierarchy, and all the three levels have the same network structure, then the attributes and behaviors of, for instance, the first level, can be inherited by the second and third levels of the hierarchy.

Firstly, the new structure has the capability of peer-to-peer communications. It needs to be emphasized that all the nodes in the new structure are homogeneous. In other words, all the nodes have the same capabilities, e.g., they have the same communication range. For example, in Figure 4.14, nodes $a$ and $b$ are neighbors in the second level, but $a$ and $b$ can not directly communicate with each other. Nodes $a$ and $b$ need to communicate via the path $(a, f, g, b)$. Similarly, nodes $v$ and $e$ can communicate via the path $(v, k, e)$. Note that Figure 4.14 only shows the first and second levels of the new structure.

Secondly, the new structure has the capability of hierarchical communications. In the new structure, the nodes in the first level and the nodes in the second level have precedence relationships. Thus the nodes at different levels
will have hierarchical communications. The precedence relationship between two nodes at two different levels is defined by the neighborhood structure of the planar graphs. As an example in Figure 4.14, node $v$ is in the second level, and nodes $f$, $g$, $h$, $i$, $j$, $k$, and $l$ are the neighbors of $v$ in the first level. Thus, there are precedence relations between $v$ and \{f, g, h, i, j, k, l\}. As in the Figure 4.14, nodes $f$ and $l$ are also neighbors of node $a$. In this case, there are also precedence relations between $a$ and \{f, l\}. This overlap precedence relations can be commonly found in the new structure, and more detail will be discussed in Chapter 5.

Figure 4.14: The new network structure has the capability of both hierarchical and peer-to-peer communications.
Chapter 5

Algorithms

In the previous chapter, the representation of change in a sensor network has been discussed. Changes of the environment are represented as sequences of snapshots taken by the sensor network, and these snapshots are decentrally stored in each sensor node. With certain spatial, temporal, and thematic granularities of observations, gradual or abrupt changes of areal objects may occur between two snapshots, such that there can be a large number of sensor nodes that need to update their local datasets. To generate qualitative or quantitative information about the areal objects, a large number of sensor nodes need to cooperate in a decentralized manner. The algorithms presented in this chapter support the communication of an arbitrarily large set of sensor nodes.

This chapter includes two major parts. The first part, including Sections 5.1 - 5.6, presents a decentralized change detection algorithm in a flat planar graph structure. In the second part, i.e., Section 5.7, a new hierarchical network structure, called multi-granularity network, is proposed. The new hierarchical structure is designed based on an object-oriented approach, such that the planar graph at each level of the hierarchy can inherit the attributes and behaviors of the already developed flat planar graph. Thus, the decentralized change detection algorithm proposed in Sections 5.1 - 5.6 can be used at each level of the hierarchical structure.

In the first part of this chapter, the decentralized change detection algorithm consists of three components, named BROADCAST, DETECT, and UPDATE respectively. The BROADCAST component will be introduced in Section 5.1. Before discussing the DETECT component, some background will be introduced in Section 5.3 and 5.4. The DETECT component is then introduced in Section 5.5. Finally, the UPDATE component is in Section 5.6.

In the previous chapter, a sensor network is modeled as a planar graph. In this chapter, for clarity reasons, a specific type of planar graph, i.e., triangula-
tion, is used. The decentralized change detection algorithm is then described using the triangulation example. Section 5.2 describes how the decentralized change detection algorithm can be adapted into a planar graph.

5.1 The BROADCAST component

In the BROADCAST component, the communication among sensor nodes is straightforward, i.e., each active node simply broadcasts its sensor reading to its immediate neighbors. The rest of the BROADCAST component does not require any other communications. The boundary edges of areal objects and the inserted and deleted edges can then be calculated locally in each sensor node using its local dataset and the information received from its immediate neighbors. Thus, the message cost in the BROADCAST component is one message per active node, and the complexity of the BROADCAST component is \(O(n)\), where \(n\) is the number of active nodes \(|active(t)|\). Note that \(active(t)\) is the set of all active nodes at time step \(t\), as introduced in Section 4.2.3. The detailed algorithms for determining boundary edges and calculating inserted and deleted edges are discussed in Section 5.1.1 and 5.1.2 respectively.

5.1.1 Boundary Edge Determination

Boundary edges of areal objects are important for the change detection algorithm. In the DETECT and UPDATE components, boundary edges will be used as communication paths among sensor nodes for data aggregation. Thus, one of the important tasks in the BROADCAST component is to determine the boundary edges of areal objects whenever areal objects have changed between two consecutive snapshots.

Object edge, object cycle, boundary edge, and non-boundary edge have been defined in Section 4.2.1. Based on these definitions, the following datasets are required in each node \(v\) for boundary edges determination. Each dataset is only required to store the data at the current time step \(t\). Histories have not been stored in these datasets.

- \(neighbor^o(v)\): the set of neighboring nodes that are located in areal objects, i.e., \(neighbor^o(v) = \{v' : v' \in neighbor(v) \text{ and } h(v', t_i) = 1\}\).
- \(edge^o(v)\): the set of object edges at the node \(v\), \(edge^o(v) = \{(v, v') : (v, v') \in E \text{ and } h(v, t_i) = h(v', t_i) = 1\}\) (Note that \((v', v)\) is similar to \((v, v')\)).
- \(cycle^o(v)\): the set of object cycles at the node \(v\), \(cycle^o(v) = \{(v, b, a, v) : h(v, t_i) = h(b, t_i) = h(a, t_i) = 1\}\).
- \(nonBoundaryEdge(v)\): the set for storing non-boundary edges at the node \(v\), \(nonBoundaryEdge(v) = \{(v, v') : (v, v') \in edge^o(v) \text{ and } (v, v') \in cycle^o(v)\}\).
Figure 5.1: Boundary edges determination. Sensor nodes located in areal objects are represented as filled squares. Empty squares are the nodes that are not in areal objects. Active nodes are marked by small circles.

As already introduced in Section 4.2.1, the set $\text{boundaryEdge}(v, t)$ is used to store boundary edges at $v$. The history of boundary edges is not required in the database, since they can be calculated using other datasets discussed in Section 4.2.2. The dataset $\text{boundaryEdge}(v, t)$ is associated with time, because boundary edges at two consecutive time steps are required to calculate deleted and inserted edges, as will be discussed in the next section.

$\text{boundaryEdge}(v, t)$: the set of boundary edges at $v$, $\text{boundaryEdge}(v, t) = \{(v, v') : (v, v') \in \text{edge}^o(v) \text{ and } (v, v') \notin \text{cycle}^o(v)\}$.

An example is provided in Figure 5.1 to illustrate how these datasets work. In the example, there is no areal object at $t_0$ as in Figure 5.1(a), and thus all defined data sets should be empty sets. An areal object appears between time step $t_0$ and $t_1$, represented as a solid area in Figure 5.1(b). Two nodes $v$ and $b$ are activated, i.e., $\text{active}(t_1) = \{b, v\}$.

An active node needs to broadcast its sensor reading to its neighbors for the construction or destruction of object edges (line 2 in Algorithm 1). For instance, the active node $v$ should report its status, i.e., $h(v, t_1) = 1$, to $a$, $b$, $c$, $d$, and $e$, which are the neighbors of $v$. Similarly, $b$ needs to send messages to its neighbors. After the reports of active nodes $v$ and $b$, the node $v$, for instance, will have one neighbor $b$ located in areal objects, i.e., $\text{neighbor}^o(v) = \{b\}$ (lines 3-5 in Algorithm 1).

Object edges are always be constructed or destructed in pairs. Two object
**Algorithm 1: Boundary edge determination**

1. **Variables:** node $v$; current time step $t_i$; previous time step $t_{i-1}$.
2. If $v \in \text{active}(t_i)$ then send $(v, h(v, t_i))$ to neighbor($v$);
3. If $v \in V$ and received $(v', h(v', t_i))$ then
   4. If $h(v', t_i) = 1$ then $\text{neighbor}^o(v) \leftarrow \text{neighbor}^o(v) \cup \{v'\}$;
   5. If $h(v', t_i) = 0$ then $\text{neighbor}^o(v) \leftarrow \text{neighbor}^o(v) - \{v'\}$;
6. If $v \in V, h(v, t_i) \land h(v', t_i) = 1$, and $\{v, v'\} \cap \text{active}(t_i) \neq \emptyset$ then
   7. $\text{edge}^o(v) \leftarrow \text{edge}^o(v) \cup \{(v, v'), (v', v)\}$;
8. If $v \in V, h(v, t_i) \land h(v', t_i) = 0$, and $\{v, v'\} \cap \text{active}(t_i) \neq \emptyset$ then
   9. $\text{edge}^o(v) \leftarrow \text{edge}^o(v) - \{(v, v'), (v', v)\}$;
10. If $v \in V, h(v, t_i) \land h(a, t_i) \land h(b, t_i) = 1$, $(v, b, a, v) \in \text{cycle}(v)$, and
    \[ \{v, a, b\} \cap \text{active}(t_i) \neq \emptyset \text{ then} \]
11. $\text{cycle}^o(v) \leftarrow \text{cycle}^o(v) \cup \{(v, b, a, v)\}$;
12. $\text{nonBoundaryEdge}(v) \leftarrow \text{nonBoundaryEdge}(v) \cup \{(v, b, a, v)\}$;
13. $\text{boundaryEdge}(v, t_i) \leftarrow \text{edge}^o(v) - \text{nonBoundaryEdge}(v)$;
14. If $v \in V, h(v, t_i) \land h(a, t_i) \land h(b, t_i) = 0$, $(v, b, a, v) \in \text{cycle}(v)$, and
    \[ \{v, a, b\} \cap \text{active}(t_i) \neq \emptyset \text{ then} \]
15. $\text{cycle}^o(v) \leftarrow \text{cycle}^o(v) - \{(v, b, a, v)\}$;
16. $\text{nonBoundaryEdge}(v) \leftarrow \text{nonBoundaryEdge}(v) - \{(v, b, a, v)\}$;
17. $\text{boundaryEdge}(v, t_i) \leftarrow \text{edge}^o(v) - \text{nonBoundaryEdge}(v)$;

edges, e.g., $(v, v')$ and $(v', v)$, will be constructed at current time $t_i$ if both $v$ and $v'$ are located in an areal object and at least one of the nodes is active node (lines 6-7 in Algorithm 1). For example in Figure 5.1(b), the node $v$ will locally construct two object edges $(v, b)$ and $(b, v)$, since $h(v, t_i) = h(b, t_i) = 1$ and $v, b \in \text{active}(t_i)$. Both $(v, b)$ and $(b, v)$ are boundary edges, because node $v$ does not detect any object cycles. The other datasets at $v$ are: $\text{cycle}^o(v) = \emptyset$, $\text{nonBoundaryEdge}(v) = \emptyset$, $\text{boundaryEdge}(v, t_i) = \{(v, b, a, v)\}$.

A node $v$ requires at least two neighbors in $\text{neighbor}^o(v)$ to check for object cycles. At time step $t_2$ in Figure 5.1(c), the areal object has expanded and $\text{active}(t_2) = \{a\}$. Thus $\text{neighbor}^o(v) = \{a, b\}$, and $v$ can locally check for object cycles (lines 10-11 in Algorithm 1). One object cycle can be found at $v$ without further communication, i.e., $\text{cycle}^o(v) = \{(v, b, a, v)\}$. Similarly, $a$ and $b$ can both locally activate the same cycle.

As soon as the object cycles $\text{cycle}^o(v)$ are found at the node $v$, the non-boundary edges and boundary edges can be determined based on $\text{cycle}^o(v)$ (lines 12-13 in Algorithm 1). For example, at time step $t_2$ the updated data sets at the node $v$ would be: $\text{cycle}^o(v) = \{(v, b, a, v)\}$, $\text{nonBoundaryEdge}^o(v) = \{(a, v), (v, b)\}$ (may not be illustrated in the figure), $\text{boundaryEdge}^o(v, t_2) = \{(v, a), (b, v)\}$.

In Figure 5.1(d), there are two active nodes $\text{active}(t_3) = \{a, b\}$, but at this
Algorithm 2: Edge insertion and deletion

1 Variables: node $v$; current time step $t_i$; previous time step $t_{i-1}$;
2 if $v \in V$ and $\text{boundaryEdge}(v, t_i) \neq \text{boundaryEdge}(v, t_{i-1})$ then
3 \hspace{1em} $\text{insertedEdge}(v, t_i) = \text{boundaryEdge}(v, t_i) - \text{boundaryEdge}(v, t_{i-1})$.
4 \hspace{1em} $\text{deletedEdge}(v, t_i) = \text{boundaryEdge}(v, t_{i-1}) - \text{boundaryEdge}(v, t_i)$.

Figure 5.2: Edge insertion and deletion. The edges at the node $v$ are highlighted. Inserted and deleted boundary edges are marked by thick solid-line arrows and thick dashed-line arrows respectively.

time step $a$ and $b$ are not located in areal objects. Similar to the construction, object edges will be destructed in pairs (line 8-9 in Algorithm 1). In this example, all the object edges will be destructed at node $v$. And the cycle $\{(v, b, a, v)\}$ will be removed from the set $\text{cycle}^a(v)$, since $h(a, t_i) = h(b, t_i) = 0$ (line 14-17 in Algorithm 1). At time step $t_3$, the datasets of $v$ are: $\text{cycle}^a(v) = \emptyset$, $\text{nonBoundaryEdge}(v) = \emptyset$, $\text{boundaryEdge}(v, t_1) = \emptyset$.

5.1.2 Inserted and Deleted Edges

To determine the inserted and deleted boundary edges at current time step $t_i$, boundary edges from two consecutive time steps $t_i$ and $t_{i-1}$ are required. In this case, a node $v$ should maintain two latest boundary edges datasets, i.e., $\text{boundaryEdge}(v, t_i)$ and $\text{boundaryEdge}(v, t_{i-1})$. As discussed in Section 4.2.2, all inserted and deleted boundary edges will be stored in the datasets $\text{insertedEdge}(v, t)$ and $\text{deletedEdge}(v, t)$ at the node $v$. Given the two datasets $\text{boundaryEdge}(v, t_i)$ and $\text{boundaryEdge}(v, t_{i-1})$, the inserted and deleted edges at time step $t_i$ can be calculated as follows:

$\text{insertedEdge}(v, t_i) = \text{boundaryEdge}(v, t_i) - \text{boundaryEdge}(v, t_{i-1})$.

$\text{deletedEdge}(v, t_i) = \text{boundaryEdge}(v, t_{i-1}) - \text{boundaryEdge}(v, t_i)$.

Figure 5.2 follows the example in Figure 5.1. The boundary edges at the node $v$ are highlighted in Figure 5.2. At the time step $t_2$, $\text{boundaryEdge}(v, t_2) = \{(v, a), (b, v)\}$, and the boundary edges at time $t_1$ is $\text{boundaryEdge}(v, t_1) = \{(v, b), (b, v)\}$. Thus, the inserted and deleted edges at the node $v$ at $t_2$ would...
be:
\[
\text{insertedEdge}(v, t_2) = \{(v, a), (b, v)\} - \{(v, b), (b, v)\} = \{(v, a)\}.
\]
\[
\text{deletedEdge}(v, t_2) = \{(v, b), (b, v)\} - \{(v, a), (b, v)\} = \{(v, b)\}.
\]
Similarly, at time step \(t_3\), \(\text{insertedEdge}(v, t_3) = \emptyset\) and \(\text{deletedEdge}(v, t_3) = \{(v, a), (b, v)\}\). The edge insertion and deletion algorithm is listed in Algorithm 2.

### 5.2 Triangulation and Planar Graph

Sections 5.1 has introduced the BROADCAST component of the decentralized change detection algorithm. The BROADCAST component is designed based on the assumption that the sensor network can be modeled as a triangulation. This section discusses how the BROADCAST component can be adapted into a planar graph.

Figure 5.3(a) and (c) show a triangulation and a planar graph respectively. In a triangulation, each cycle has exactly three vertices or nodes, e.g., \((v, b, a, v)\) in Figure 5.3(a). While, in a planar graph, a cycle can have four or more vertices, e.g., \((v, c, h, g, b, v)\) in Figure 5.3(c).

![Figure 5.3: (a)-(b) A triangulation. (c)-(d) A planar graph.](image)

In the BROADCAST component, object cycles need to be firstly detected, so that non-boundary edges and boundary edges can be determined based on the object cycles. The inserted and deleted edges can then be further calculated using boundary edges from two consecutive time steps. Since a triangulation of sensor nodes is assumed, each active node only broadcasts its sensor reading to
its immediate neighbors, and the object cycles can be detected without further communications.

However, in the case of a planar graph, if each active node only broadcasts to its immediate neighbors, the object cycles may not be detected. For example, in Figure 5.3(c)-(d), an areal object appears between \( t_1 \) and \( t_2 \), and there are four active nodes \( v, a, f, \) and \( e \) at \( t_2 \). The active nodes \( v, a, f, \) and \( e \) will broadcast to their neighbors \( \{..., e, a, ...,\} \), \( \{..., v, f, ...,\} \), \( \{..., a, e, ...,\} \), and \( \{..., f, v, ...\} \) respectively. Thus, the node \( v \), for example, does not know the sensor reading of the node \( f \). Consequently, the node \( v \) is not able to detect the object cycle \((v, a, f, e, v)\) at \( t_2 \).

To detect object cycles in a planar graph, an active node \( v \) must send its sensor reading to all the vertices of the cycles that the active node \( v \) belongs to. For example, in Figure 5.3(d), the active node \( f \) belongs to two cycles \((f, e, v, a, f)\) and \((f, i, e, f)\), so \( f \) needs to send its sensor reading to all the vertices, i.e., \( e, v, a, \) and \( i \), of the two cycles.

As a summary, in Algorithm 1, the communications among sensor nodes (Algorithm 1, line 2) needs to be changed accordingly for planar graphs. The calculation of boundary edges (Algorithm 1, line 3-17) does not need to be changed.

The BROADCAST component calculates the inserted and deleted edges, while the DETECT and UPDATE components only use the inserted and deleted edges. Thus, the DETECT and UPDATE components are independent to triangulation and planar graph structures.

### 5.3 Trajectory

The BROADCAST component discussed in Section 5.1 efficiently uses one message per active node and let the sensor network be ready for change detection in the DETECT component. Before the introduction of the DETECT component in Section 5.5, some new concepts are first introduced in this section.

#### 5.3.1 Leader and Trajectory

In the proposed sensor network model in Chapter 4, spatiotemporal data is decentrally stored in each node in the network. These spatiotemporal data acquired at different time steps may need to be accessed for qualitative and quantitative analysis. For example, two consecutive snapshots at time steps \( t_i \) and \( t_{i-1} \) could be used to calculate the area difference of an areal object between the two time steps. This section introduces the new concept of leader and trajectory, which will enable the decentralized retrieval of historical spatiotemporal
A leader of an areal object is defined as a selected node that is currently located inside the areal object. Since areal objects are changing over time, the leader needs to be dynamically updated. A leader trajectory, or simply trajectory, is the path of the leader over time. A trajectory consists of a sequence of edges, called trajectory edges. Figure 5.4(a)-(f) show a general idea about the update of leaders and leader trajectories. Between time step $t_0$ and $t_1$, an areal object appears. A node $a$ on the boundary edges of the areal object is selected as the leader. The leader $a$ reports the appearance of the areal object to the sink by greedy routing (refer to [75, 77]). During the routing, the information of the leader $a$ will be recorded along the path from $a$ to the sink. The path between the leader $a$ and the sink is regarded as one part of the leader trajectory. As in Figure 5.4(c), the areal object contracts between $t_1$ and $t_2$, and the leader $a$ is not longer in the areal object. The node $b$ is then selected as the new leader, and the path between the two leaders $a$ and $b$ is added into the leader trajectory. At $t_3$ in Figure 5.4(d), although the areal object has expanded, the leader $b$ is still on the boundary edges. Thus, no update of the trajectory is required. Similar to $t_2$, there is an update of the trajectory between leader $b$ and $c$ at $t_4$.

Finally, between $t_4$ and $t_5$, the areal object disappears. The leader $c$ will report the disappearance of the areal object to the sink. And at the same time,
Figure 5.5: Trajectory Identity. Boundary edges of the areal object have the same identity as the trajectory edges.

To correctly access spatiotemporal data of areal objects, each leader trajectory must have an identity. The identity of a trajectory will be decentrally stored in each trajectory edge. For example, in Figure 5.5(a), node a is a leader and a trajectory identity, e.g., A, is given to each edge on the trajectory. To correctly access each snapshot of an areal object from the trajectory, each boundary edge of the areal object at each snapshot should also have an identity. And the identity of boundary edges should match the identity of the trajectory. In Figure 5.5(a), each boundary edge of the areal object is also provided the same identity A.

The identity A will be used to identify the trajectory over time. Thus, when the areal object changes over time, the same identity should be maintained. In Figure 5.5(b), the areal object has contracted at t2, and the leader is changed from a to b. The same identity A should be updated for both new trajectory edges and new boundary edges, as shown in Figure 5.5(b).

As discussed in Section 4.2, each boundary edge of areal objects has a timestamp. For instance, the boundary edges in Figure 5.5 (a) and (b) are associated with a timestamp of t1 and t2 respectively. The snapshot sequences of areal objects can be easily retrieved using the trajectory and its identities. For example, if a query inserted from the sink is interested in the areal objects at t2, then the relevant data can be retrieved by a traversal from the sink to the leader b. And the leader b (with identity A) can then traverse all the boundary edges that have a timestamp of t2 and have the same identity of A.
can be used for qualitative and quantitative analysis, the trajectory identities need to firstly be maintained when areal objects change over time. The detail approach for trajectory identity maintenance is discussed in the next section.

5.3.2 Trajectory Maintenance

There are eight types of spatial changes of areal objects that require possible updates of trajectories. These eight types of changes include six topological changes and two non-topological changes, as already introduced in Section 3.5.

When an areal object appears, a new trajectory needs to be created for the areal object, and the trajectory needs a unique identity. Figure 5.6 has an abstract representation of the changes of trajectories. In the figure, the star represents the sink, circles or nodes represent leaders, and a solid line represents a path between two nodes. In Figure 5.6(a)-(b), an areal object appears, a leader is selected, and an identity of $A$ is given to the leader. This thesis assumes that a unique trajectory identity can always be generated locally in a sensor node based on, for instance, the combination of the node identity and the time stamp. Also the path between the leader and the sink has the same identity $A$. An areal object with a trajectory identity $A$ will be simply called areal object $A$.

As discussed in the previous section, when an areal object expands or contracts, the trajectory of the areal object may need to be updated. In Figure 5.6(c), a new leader is selected for the trajectory $A$ due to the expansion of the areal object, and there is a path between the two leaders. Between $t_2$ and $t_3$, areal object $A$ contracts and a new areal object $B$ appears, and their trajectories are updated correspondingly.

If two areal objects $A$ and $B$ merge, the two trajectories of the areal objects have to merge into one trajectory, as in Figure 5.6(e). One identity, e.g., $A$, can remain for the merged trajectory, and the other trajectory $B$ is concluded to the sink. With the conclusion of trajectory $B$, the sink will also aware the merge of two areal objects $A$ and $B$. In contrast, if an areal object $A$ splits into two areal objects, the trajectory $A$ has to split into two trajectories. In Figure 5.6(f), the areal object $A$ may have some expansion at $t_5$, and at $t_6$ areal object $A$ has split into two areal objects. One of the areal object can use the existing trajectory $A$, and the other should have a new trajectory, e.g., $C$. The new trajectory $C$ should establish a path to the sink, and the sink will also aware the split of the areal object $A$. The areal objects $A$ and $C$ may have contracted at $t_7$ in Figure 5.6(g), and at $t_8$ both areal objects $A$ and $C$ has disappeared. Both trajectories $A$ and $C$ conclude to the sink.

The above discussion does not include areal objects with holes. The maintenance of trajectories in the case of regions with a hole is shown in Figure 5.7. At
Figure 5.6: The trajectories need to be updated when areal objects change over time. The star represents the sink, different circles or nodes at one time step represent different leaders, and a solid line represent a path between two nodes. There may be multiple stars at one time step, but these stars represent the same sink at a single location.

Figure 5.7: Trajectory maintenance for regions with a hole. A region with a hole is specified with superscript −, e.g., \( B^- \).

\( t_1 \), there is an areal object \( A \) as in Figure 5.7(a), and between \( t_1 \) and \( t_2 \) a hole appears inside the areal object \( A \). When a hole appears, a new trajectory will be created for the hole. In Figure 5.7(b), the hole is provided a trajectory identity of \( B^- \), where the superscript − specifies that the spatial object is a hole. If there is a partial-split of the areal object \( A \), the hole \( B^- \) will disappear. The two trajectories \( A \) and \( B^- \) will merge into one trajectory \( A \), and the trajectory of the hole \( B^- \) is concluded to the sink, as shown in Figure 5.7(c). If the areal object \( A \) self-merges between Figure 5.7 (c)-(d), a hole will be formed inside the areal object, and the trajectory \( A \) will split into two trajectories \( A \) and \( C^- \).

Trajectories need to be updated differently when areal objects undergo different types of changes over time. The DETECT component should firstly detect and distinguish different types of changes, so that trajectories can be updated properly in the UPDATE component. A large number of nodes may be involved in the changes of areal objects in the cases of abrupt change, and thus
in-network aggregations among relevant nodes are necessary. In the DETECT component, graph traversals are used to organize the aggregation of decentralized spatiotemporal data for change detection. The next section discusses how graph traversals can be used to detect different types of changes.

5.4 Change Detection

As introduced in Section 3.5, there are eight types of changes of areal objects. Different changes will have different types of graph traversals, and all of the traversals will form closed traversable trails. This section discusses these changes in four groups: appearance and disappearance, expansion and contraction, merge and split, self-merge and partial-split.

5.4.1 Appearance and Disappearance

There is a closed traversable trail for the appearance of areal objects. In Figure 5.8(a), an areal object appears, and a set of boundary edges have been inserted. The set of inserted boundary edges forms a closed trail. Suppose the traversal starts at node $i$, then the trail is: $i \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i$. An abstract representation of the traversal in Figure 5.8(a) is shown in (b), where the solid-line arrow represents a traversal on inserted edges, and the circle represents a node that has a node identity of $i$. Note that in Figure 5.8(a), the trajectory identity has not been added to the inserted edges. All the figures in Section 5.4 and 5.5 illustrate the stage when changes of areal objects have occurred but trajectory identities have not been updated. The update of trajectory identities will be discussed in the UPDATE component in Section 5.6.

In Figure 5.8(c), an areal object disappears, and all the boundary edges with trajectory identity of $A$ have been deleted. And in Figure 5.8(d), the dashed-line arrow denotes a traversal on deleted edges. In Figure 5.8(b) and (d), each of the two traversals has only one segment, and the segment starts at node $i$ and ends at node $i$. Node $i$ is called the KEY node of the traversal.

5.4.2 Expansion and Contraction

For the expansion and contraction of areal objects, the closed trail will include two segments. One segment of the traversal consists of inserted edges, and another segment consists of deleted edges. In Figure 5.9(a), for example, an areal object expands, and there are six inserted edges and two deleted edges. The inserted edges form one segment of the traversal: $b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h$. The deleted edges form another segment: $h \rightarrow j \rightarrow b$. Notice that the second
Figure 5.8: Appearance and disappearance. (a) an areal object appears. (b) is an abstract representation of the traversal in (a). (c) an areal object disappears. (d) represents the traversal in (c). Note that the node identities, i.e., a and b, is in lower cases, and trajectory identities, i.e., A, is in upper cases.

segment of the traversal is in reverse direction. If the traversal starts at node b, the traversal can then return to b. Figure 5.9(b) illustrates that the two segments of the traversal, i.e., 1 and 2, are connected by two nodes b and h. The two nodes b and h are the KEY nodes of the traversal. Traversal segment 2 aware the trajectory identity A, while traversal segment 1 still do not have trajectory identity.

Similarly, in Figure 5.9(c) and (d), an areal object contracts, and there are two segments of traversals that are connected by two KEY nodes b and h. Section 5.5 will demonstrate how expansion and contraction can be distinguished: the traversal of expansion consists of (1) inserted edges and (2) deleted edges, and the traversal of contraction consists of (1) deleted edges and (2) inserted edges.

One special case of expansion is shown in Figure 5.10, where there is only one node e located in the areal object at t1, and thus there are no boundary edges. The areal object has expanded at t2, and there are five inserted boundary edges, but there are no deleted edges. The inserted edges form a closed trail: $e \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$. In this case, the traversal of expansion only includes one segment. This special case can exist in other types of changes as well, and the DETECT component is able to handle these cases, as will be discussed in Section 5.5.1.

Note that trajectory identities are always stored at boundary edges. The only case that the trajectory identity is stored at a node is shown in Figure 5.10. At
Figure 5.9: Expansion and contraction. The traversal for expansion or contraction has two segments, 1 and 2, and two KEY nodes, b and h, as in (b) and (d).

Figure 5.10: A special case of expansion, in which the traversal only has one segment. This special case can exist in other types of changes as well.

5.4.3 Merge and Split

The closed trail for the merge of two areal objects includes four segments. Figure 5.11(a) shows a merge of two areal objects \( A \) and \( B \), and Figure 5.11(b) shows that the traversal starts from node \( a \). The first segment of the traversal is on inserted edges from node \( a \) to node \( d \): \( a \rightarrow b \rightarrow c \rightarrow d \). Two KEY nodes \( a \) and \( d \) are located in different areal objects, i.e., \( A \) and \( B \) respectively. The second segment of the traversal is on the deleted edges of areal object \( B \) from node \( d \) to node \( m \): \( d \rightarrow e \rightarrow m \), and the deleted edges have the trajectory identity \( B \), as shown in Figure 5.11(a) and (b). The third segment will start from areal object \( B \) and return to areal object \( A \) via inserted edges: \( m \rightarrow f \rightarrow g \). Finally, the fourth segment of the traversal will return to the original node \( a \) via the deleted edges.
edges of areal object A: g → j → a. Note that traversal segments 2 and 4 are in reverse direction. These four segments of traversal are connected by four KEY nodes a, d, m, and g, in which two nodes are located in areal object A, and the other two are in areal object B.

The traversal of a split also includes four segments, and the four segments are in the order of (1) deleted edges and (2) inserted edges and (3) deleted edges and (4) inserted edges, as in Figure 5.11(c) and (d). The four segments of traversal are also connected by four KEY nodes a, d, m, and g. The traversal segment 1 and 3 on deleted edges aware the trajectory identity A. In comparison, the four traversal segments of a merge are in a different order: (1) inserted edges and (2) deleted edges and (3) inserted edges and (4) deleted edges, as in Figure 5.11(b). Both of the traversals in Figure 5.11(b) and (d) start at node a. The selection of the traversal starting node will be discussed in Section 5.5.

5.4.4 Self-merge and Partial-split

In Figure 5.12(a), an areal object A self-merges into a region with a hole. The traversal of self-merge has a similar structure to a merge: (1) inserted edges and (2) deleted edges and (3) inserted edges and (4) deleted edges, as shown in Figure 5.12(b). There are four KEY nodes j, d, l, and g that connect the four segments of the traversal. The segment 2 and 4 of a self-merge have the same

Figure 5.11: Merge and split. The traversals for merge and split include four segments. Two segments of the traversal are on inserted edges, and the other two segments are on deleted edges, as in (b) and (d). The traversals start at node a.
Figure 5.12: The traversals for self-merge and partial-split include four segments and four KEY nodes.

trajectory identity $A$, as illustrated in Figure 5.12(a) and (b). In comparison, section 2 and 4 of a merge in Figure 5.11(a) and (b) have different trajectory identities, i.e., $B$ and $A$ respectively. Thus, merges and self-merges can be distinguished by the trajectory identities of the traversal segments, as in Figure 5.11(b) and Figure 5.12(b).

The first and third segments of the self-merge traversal consist of inserted edges: one would belong to the internal boundary edges of the hole, i.e., $j \rightarrow n \rightarrow e \rightarrow d$, and the other would belong to the external boundary edges of the areal object, i.e., $l \rightarrow m \rightarrow f \rightarrow g$. The second and fourth segments of the traversal, i.e., $d \rightarrow k \rightarrow l$ and $g \rightarrow j$ consist of deleted edges, and they are in the reverse direction.

In Figure 5.12(c), an areal object with a hole partial-splits. The areal object with a hole consists of two closed trails: one is internal for the hole and the other is external for the areal object, as already introduced in Section 4.1.3. As in Section 5.3.2, the trajectory identity for a hole of an areal object will be specified with a superscript $\sim$. Figure 5.12(c) shows that all the internal edges with $B^-$ form a closed trail: $b \rightarrow j \rightarrow n \rightarrow e \rightarrow d \rightarrow c \rightarrow b$. Similar to splits, the traversal for partial-split has four segments, and the order is (1) deleted edges and (2) inserted edges and (3) deleted edges and (4) inserted edges. For a split, segments 1 and 3, in Figure 5.11(d), have the same trajectory identity $A$, while in the case of partial-split in Figure 5.12(d), segment 1 has the trajectory identity of the hole $B^-$ and segment 3 has a different trajectory identity $A$. 
5.4.5 Summary

As a summary, there are eight different types of basic traversals on deleted and inserted edges. These eight traversals can be distinguished based on their different segments. Each type of traversal can uniquely identify one type of change. Thus, the six topological changes and two non-topological changes can be identified by distinguishing the different types of traversals, as shown in Figure 5.13(a)-(h).

There are possible other types of traversals, and they can be considered as the combinations of the eight basic traversal types. For example, in Figure 5.14, an areal object expands, and the deleted and inserted edges have been separated into two closed trails, i.e., $d \rightarrow o \rightarrow n \rightarrow d$ and $i \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i$. The traversals in Figure 5.14 can be considered as a combination of the two traversals in Figure 5.13(a) and (b). Chapters 7 and 8 will further discuss the combinations of traversals.

5.5 The DETECT Component

As discussed in Section 5.4, there are eight different types of traversals that can be used to distinguish eight types of changes. The DETECT component in this section will provide decentralized algorithms for traversal organization and change detection. The basic idea in the DETECT component is to initialize a traversal at a given node $v$ with a message, and the message will be passed from one node to another during the traversal. Also, decentralized data will
be aggregated into the message during the traversal. Since all the traversals form closed trails, the aggregated message will return to its origin node \( v \), such that node \( v \) will be able to detect different types of changes based on the aggregated message. Before the discussion of the detailed algorithm, the algorithm preliminaries are firstly introduced.

### 5.5.1 Preliminaries

Three parts can be found in this section, i.e., messages, absent traversal segments, and transition edges, respectively. The DETECT component includes two types of traversal organization approaches to initialize and conclude traversals. As will be discussed in the following sections, one type of traversal organization approach uses transition edges for traversal initialization, while another type of approach does not use transition edges.

#### Messages

Since trajectory identities of traversal segments can be used to identity different types of changes as in Figure 5.13, relevant trajectory identities will be aggregated during the traversal. A message will be initialized at the beginning of a traversal for data aggregation in the network. A message initialized at a node \( v \) is denoted as follows:

\[
\text{msg}(v) = (n_1, p_1, n_2, p_2, n_3, p_3, n_4, p_4).
\]

There are eight elements in the message: \( n_1, n_2, n_3, \text{ and } n_4 \) denote the node identity of the four KEY nodes, and \( p_1, p_2, p_3, \text{ and } p_4 \) denote the trajectory identity of the four segments in a traversal. In the algorithm, \( \text{msg}(v).n_1 \), for example, is used to represent the element \( n_1 \) in the message \( \text{msg}(v) \). In the beginning of a traversal, all the elements in a message are empty, and the message can be represented as:
\[ \text{msg}(v) = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset). \]

The KEY nodes are responsible to update the elements of a message during a traversal. For example, the closed trail for a split in Figure 5.17(d) consists of four KEY nodes, i.e., \(c, k, e, \) and \(b,\) and four segments, i.e., \(c \rightarrow d \rightarrow k, k \rightarrow e, e \rightarrow v \rightarrow b, \) and \(b \rightarrow c.\) If node \(c\) initializes a traversal with a message \(\text{msg}(c),\) then node \(c\) will update the elements \(\text{msg}(c).n_1\) and \(\text{msg}(c).p_1,\) node \(k\) will update \(\text{msg}(c).n_2\) and \(\text{msg}(c).p_2,\) node \(e\) will update \(\text{msg}(c).n_3\) and \(\text{msg}(c).p_3,\) and node \(b\) will update \(\text{msg}(c).n_4\) and \(\text{msg}(c).p_4.\) Finally, the traversal will return to the initial node \(c\) with a message that contains all the information about the closed trail.

The node identity of a node \(v\) is simply represented as \(v,\) i.e., \(\text{msg}(c).n_1 = c, \text{msg}(c).n_2 = k, \text{msg}(c).n_3 = e, \) and \(\text{msg}(c).n_4 = b.\) All the edges in the same segment have the same trajectory identity. For example, in Figure 5.17(d), the edges \((c, d)\) and \((d, k)\) in the first segment have the same identity of \(A.\) In the algorithm, \(id_j(v, v', t)\) is used to represent the trajectory identity of an edge \((v, v')\) at time \(t.\) For example, in Figure 5.17(d), \(id_j(c, d, t_1) = A\) and thus \(\text{msg}(c).p_1 = id_j(c, d, t_1) = A.\) Similarly, \(\text{msg}(c).p_2 = I, \text{msg}(c).p_3 = A,\) and \(\text{msg}(c).p_4 = I.\) Note that \(\text{msg}(c).p_2 = I \) and \(\text{msg}(c).p_4 = I\) are used to specify that the second and fourth segments of the traversal are on inserted edges and trajectory identities have not been provided. In Figure 5.17(d), a completed message for the traversal started at node \(c\) would be:

\[ \text{msg}(c) = (c, A, k, I, e, A, b, I). \]

Similarly, in Figure 5.17(b), two areal objects have merged, and a completed message for a traversal started at \(c\) would be:

\[ \text{msg}(c) = (c, I, k, B, e, I, b, A). \]

In Figure 5.16(b), an areal object has expanded. If a traversal is initialized at node \(c\) with a message \(\text{msg}(c),\) then the completed message would be:

\[ \text{msg}(c) = (c, I, a, A, \emptyset, \emptyset, \emptyset, \emptyset). \]

Note that \(\text{msg}(c).n_3, \text{msg}(c).p_3, \text{msg}(c).n_4,\) and \(\text{msg}(c).p_4\) are all empty, since a traversal for expansion has only two segments and two KEY nodes.

**Absent Traversal Segments**

There are four KEY nodes and four traversal segments in both the examples in Figure 5.17(b) and (d). Each segment is a trail that starts at a KEY node and ends at another KEY node, and the trail consists of at least one edge.
Figure 5.15: It is possible that there are absent segments in a traversal. KEY nodes are marked by circles. (a) A segment of deleted edges is absent at KEY node \( b \). (b) A segment of deleted edges is absent at KEY node \( d \). (c) Two segments of inserted edges are absent at KEY nodes \( b \) and \( d \).

In some traversals, it is possible that there is no edge in a segment, and the segment is regarded as an \textit{absent} segment. For example, in Figure 5.15(b), if node \( b \) start a traversal, then the traversal would only contain three segments: \( b \to c \to d \) on inserted edges, \( d \to c \to a \) on inserted edges, and \( a \to b \) on a deleted edge. Since a traversal of merge has \( 1 \) \textit{inserted edges} and \( 2 \) \textit{deleted edges} and \( 3 \) \textit{inserted edges} and \( 4 \) \textit{deleted edges}, as in Section 5.4.3, the second segment on deleted edges is regarded as an absent segment at node \( d \).

Generally, in the traversals of merge, split, self-merge and partial-split, the first and third segments always exist, while the second and fourth segments may be absent. In the traversals for expansion and contraction, the first segment always exists, while the second segment may be absent, as an example in Figure 5.15(a).

In the DETECT component, an absent segment at a KEY node \( v \) is considered as a segment that starts at \( v \) and ends at \( v \). And the trajectory identity of the absent segment denotes as \( id_j(v, v, t) \). In Figure 5.15(b), for example, the completed message initialized by node \( b \) would be:

\[
\text{msg}(b) = (b, I, d, B, d, I, a, A).
\]

In the message, \( \text{msg}(b).n_2 = \text{msg}(b).n_3 = d \), and \( \text{msg}(b).p_2 = B \) represent that the second segment is an absent segment at KEY node \( d \), and the segment has a trajectory identity of \( B \). The trajectory identity for the second segment
should be provided by node \(d\). As shown in Figure 5.15(b), node \(d\) can acquire the trajectory identity \(B\) from its boundary edges \{(\(d, e\), \(e, d\))\}.

Another example of absent segments is shown in Figure 5.15(c), where an areal object has split. The traversal of a split is in the order of (1) deleted edges and (2) inserted edges and (3) deleted edges and (4) inserted edges. If node \(b\) starts a traversal, then the first and third segments, i.e., \(b \rightarrow c \rightarrow d\) and \(d \rightarrow a \rightarrow b\), are both deleted edges. The second and fourth segments of inserted edges are absent at KEY nodes \(d\) and \(b\) respectively. Thus, the completed message created by node \(b\) would be:

\[msg(c) = (b, A, d, I, d, A, b, I).\]

Since the traversal forms a closed trail, thus \(msg(c).n_4 = msg(c).n_1 = b\), and \(msg(c).p_4 = I\) represent that the fourth segment of the traversal is an absent segment at node \(b\).

**Transition Edges**

Since the DETECT component should be able to efficiently organize traversals for decentralized data aggregation, only a small subset of sensor nodes should be nominated for initializing traversals. As introduced in the beginning of Section 5.5.1, transition edges are important for traversal organization. A transition edge is defined as below:

**Definition 14** An inserted or deleted edge is a transition edge if the edge connects an active node and a non-active node.

For example, in Figure 5.16(b), the inserted edge \((c, d)\) is a transition edge that has a initial node \(c \notin active(t_2)\) and a terminal node \(d \in active(t_2)\). Also \((e, a)\) is a transition edge, but the edge begins at an active node \(e\) and ends at a non-active node \(a\).

Transition edges are always in pairs. If there is a transition edge with an initial non-active node and a terminal active node, then there always exists another transition edge with an initial active node and a terminal non-active node. A pair of transition edges is always connected by a trail. For example, in Figure 5.16(b), \((c, d)\) and \((e, a)\) are connected by the trail \(c \rightarrow d \rightarrow e \rightarrow a\). The set of transition edges of a node \(v\) is denoted as \(transitionEdge(v, t)\), for instance, \(transitionEdge(c, t_2) = \{(c, d)\}\). Three different types of nodes in a sensor network can then be distinguished:

Type 1). A node \(v\) that is a non-active node at \(t\) and is the initial node of transition edges at \(t\), i.e., \(v \notin active(t)\) and \(\exists(a, b) \in transitionEdge(v, t), a = v\). An example is \(c\) in Figure 5.16(b).
Figure 5.16: Expansion and contraction. There are two active nodes $d$ and $e$ in both expansion and contraction. Transition edges are marked by dashed-line ellipses.

**Type 2.** A node $v$ that is a non-active node at $t$ and is the terminal node of transition edges at $t$, i.e., $v \notin \text{active}(t)$ and $\exists (a, b) \in \text{transitionEdge}(v, t)$, $b = v$. An example is $a$ in Figure 5.16(b).

**Type 3.** A node $v$ that is neither a type 1 node nor a type 2 node at $t$, and has inserted and/or deleted edges at $t$, i.e., $\text{insertedEdge}(v, t) \cup \text{deletedEdge}(v, t) \neq \emptyset$. Examples are $d$, $e$, and $v$ in Figure 5.16(b).

It is possible that there are multiple type 1 and type 2 nodes in a closed trail. For example, in Figure 5.17(d), there are four transition edges: $(c, d)$, $(d, k)$, $(e, v)$, and $(v, b)$. Thus, there are two type 1 nodes $c$ and $e$, and two type 2 nodes $k$ and $b$.

In the following two sections, two types of traversal organization approaches will be introduced. The first approach requires transition edges, while the second approach does not require transition edges.

### 5.5.2 Traversal Approach 1

Since the first traversal organization approach is based on transition edges, the approach will be organized into three parts based on the three different types of nodes defined in the previous section. In the following, the decentralized algorithms for the three different types of nodes will be introduced respectively.
Initial Non-active Nodes of Transition Edges

If a non-active node $v$ is the initial node of a transition edge, then it is a type 1 node. A type 1 node has three responsibilities. Firstly, a type 1 node needs to initialize a traversal with a message. Secondly, a type 1 node should check the traversals from other type 1 nodes. And thirdly, a type 1 node should conclude its own traversal. These three responsibilities are discussed below respectively.

Firstly, a type 1 node can be determined in Algorithm 3, lines 2-3. If a node $v$ is a type 1 node, it should create a new message (Algorithm 3, line 6), add its local data into the message (Algorithm 3, lines 7-8), and send the message to the next node on the traversal (Algorithm 3, line 9).

For example, in Figure 5.17(d), node $c$ is a non-active node, and it is the initial node of a transition edge $(c,d)$, so $c$ is a type 1 node. Node $c$ should create a new message $msg(c)$, and update elements $n_1$ and $p_1$ of the message, i.e., $msg(c).n_1 = c$, and $msg(c).p_1 = id_j(c,d,t_1) = A$. Node $c$ should then send the message $msg(c) = (c, A, \emptyset, \emptyset, \emptyset, \emptyset)$ to the next node $d$. Similarly, in Figure 5.17(d), node $c$ is also a type 1 node, and it should send a message $msg(c)$ to its next node $e$.

Secondly, a type 1 node $v$ can receive a message $msg(v')$ that is initialized by another type 1 node $v'$ (Algorithm 3, lines 11-26). As discussed in Section 5.5.1, there can be multiple type 1 nodes in a closed trail, therefore multiple traversals
Algorithm 3: Initial Non-active Nodes of Transition Edges

1. **Variables:** node $v$; current time step $t_i$; previous time step $t_{i-1}$;
2. if $v \notin \text{active}(t_i)$ and $\exists (a, b) \in \text{transitionEdge}(v, t_i)$, $a = v$ then
   3. $v$ is set as type 1 node at $t_i$;
4. if $v$ is a type 1 node at $t_i$ then
   5. $v$ find the terminal node $v_{\text{term}}$ on the transition edge;
   6. $v$ create $\text{msg}(v)$;
   7. $\text{msg}(v).n_1 \leftarrow v$;
   8. $\text{msg}(v).p_1 \leftarrow \text{id}_j(v, v_{\text{term}}, t_{i-1})$;
   9. $v$ send $\text{msg}(v)$ to $v_{\text{term}}$;
10. if $v$ is a type 1 node at $t_i$ and receive $\text{msg}(v')$ from $v''$ then
    11. if $v \neq v'$ then
        12. if $v < v'$ then
            13. the traversal started from $v'$ is terminated at $v$;
        14. else if $v > v'$ then
            15. if $\text{msg}(v').p_1 \neq \emptyset$ and $\text{msg}(v').p_2 = \emptyset$ then
                16. $v$ find the terminal node $v_{\text{term}}$ on the transition edge;
                17. $\text{msg}(v').n_2 \leftarrow v$;
                18. $\text{msg}(v').p_2 \leftarrow \text{id}_j(v, v, t_{i-1})$;
                19. $\text{msg}(v').n_3 \leftarrow v$;
                20. $\text{msg}(v').p_3 \leftarrow \text{id}_j(v, v_{\text{term}}, t_{i-1})$;
                21. $v$ send $\text{msg}(v')$ to $v_{\text{term}}$;
            22. else if $\text{msg}(v').p_2 \neq \emptyset$ and $\text{msg}(v').p_3 = \emptyset$ then
                23. $v$ find the terminal node $v_{\text{term}}$ on the transition edge;
                24. $\text{msg}(v').n_3 \leftarrow v$;
                25. $\text{msg}(v').p_3 \leftarrow \text{id}_j(v, v_{\text{term}}, t_{i-1})$;
                26. $v$ send $\text{msg}(v')$ to $v_{\text{term}}$;
        27. else if $v = v'$ then
            28. if $\text{msg}(v').p_1 \neq \emptyset$ and $\text{msg}(v').p_2 = \emptyset$ then
                29. $\text{msg}(v').n_2 \leftarrow v$;
                30. $\text{msg}(v').p_2 \leftarrow \text{id}_j(v, v, t_{i-1})$;
                31. the traversal is closed at $v$; $v$ detects change using Algorithm 7;
            32. else if $\text{msg}(v').p_2 \neq \emptyset$ and $\text{msg}(v').p_3 = \emptyset$ then
                33. the traversal is closed at $v$; $v$ detects change using Algorithm 7;
            34. else if $\text{msg}(v').p_1 \neq \emptyset$ and $\text{msg}(v').p_4 = \emptyset$ then
                35. $\text{msg}(v').n_4 \leftarrow v$;
                36. $\text{msg}(v').p_4 \leftarrow \text{id}_j(v, v, t_{i-1})$;
                37. the traversal is closed at $v$; $v$ detects change using Algorithm 7;
            38. else if $\text{msg}(v').p_4 \neq \emptyset$ then
                39. the traversal is closed at $v$; $v$ detects change using Algorithm 7;
can be initialized locally by these type 1 nodes. For example, in Figure 5.17(d), both type 1 nodes c and e will initialize traversals.

To reduce message costs, the two nodes c and e will compare their privilege values, such that only one traversal remains. In this algorithm, the node identities are used as privilege values. If a type 1 node v′ has a larger node identity than another type 1 node v, the traversal from v′ will be terminated at v (Algorithm 3, lines 12-13). On the other hand, if v′ has a smaller node identity than v, node v will continue the traversal of v′ (Algorithm 3, lines 14-26). In the example in Figure 5.17(d), since e > c, the traversal started at node e will be terminated by node c, i.e., e → v → b → c. While node e will continue the traversal started at node c, such that the traversal can return to the initial node c, i.e., c → d → k → e → v → b → c.

Before a type 1 node v continues the traversal initialized by another type 1 node v′, node v should update msg(v′). The message msg(v′) that node v receives could have two different cases: (a) msg(v′).p1 ≠ ∅ and msg(v′).p2 = ∅ and (b) msg(v′).p2 ≠ ∅ and msg(v′).p3 = ∅.

The first case implies that node v, which receives msg(v′), is the second KEY node on the traversal, since msg(v′).p2 = ∅ (Algorithm 3, lines 15-21). An example is provided in Figure 5.15(b), where there are two type 1 nodes b and d. A traversal will be initialized by node b, and node d will be the second transition node on the traversal.

If a type 1 node v receives a message msg(v′), in which msg(v′).p2 ≠ ∅ and msg(v′).p3 = ∅, then node v will be the third KEY node of a traversal. As discussed above, in Figure 5.17(d), both node c and e will start a traversal. Node c will receive a message msg(c) = (c, A, k, I, ∅, ∅, ∅) from k. Since e is the third KEY node, node e will update the message (Algorithm 3, lines 22-26) and send an updated message to the next node v: msg(c) = (c, A, k, I, e, A, ∅, ∅). Similarly, node c will receive a message msg(e) = (e, A, b, I, ∅, ∅, ∅, ∅) from b. Since c < e, node c will terminate the traversal from e, and the message msg(e) will be deleted by c.

Thirdly, a type 1 node v can receive a message msg(v′) that is initialized by itself, i.e., v = v′. As in Algorithm 3 (lines 27-39), the message msg(v′) that node v receives can have four different cases: (a) msg(v′).p1 ≠ ∅ and msg(v′).p2 = ∅, (b) msg(v′).p2 ≠ ∅ and msg(v′).p3 = ∅, (c) msg(v′).p3 ≠ ∅ and msg(v′).p4 = ∅, and (d) msg(v′).p4 ≠ ∅.

The first case is shown in Figure 5.15(a), where node b is the type 1 node, and it should start a traversal with a message msg(b). The first segment of the traversal will be: b → c → d → a → b. Node b would then receive msg(b) = (b, I, ∅, ∅, ∅, ∅, ∅, ∅) from node a. The second segment is absent at b. Node b should update the message and close the traversal started by itself, as in
Algorithm 3 (lines 28-31).

The second case can be found in Figure 5.16(b), where the only traversal will start at node c with \( \text{msg}(c) = (c, I, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \). The first and second segments of the traversal are \( c \rightarrow d \rightarrow e \rightarrow a \) and \( a \rightarrow v \rightarrow c \). Node c would receive \( \text{msg}(c) = (c, I, a, A, \emptyset, \emptyset, \emptyset) \) from node v, i.e., \( \text{msg}(c).p_2 = A \neq \emptyset \) and \( \text{msg}(c).p_3 = \emptyset \). No further update of the message is required, and node c will simply close the traversal (Algorithm 3, lines 32-33).

An example of the third case is in Figure 5.15(c), where the traversal started at node b will traverse the first and third segments, i.e., \( b \rightarrow c \rightarrow d \rightarrow a \rightarrow b \), and return to the starting node b. Notice that the second and fourth segments of the traversal are absent. Node b would receive a message \( \text{msg}(b) = (b, A, d, I, d, I, a, A) \) from node a. In the message, \( \text{msg}(b).p_4 = A \neq \emptyset \). Node b should update the absent fourth segment and finish its own traversal (Algorithm 3, lines 34-37).

Finally, an example of the fourth case is in Figure 5.15(b). In the figure, node b will start a traversal, and the first, third and fourth segments of the traversal are \( b \rightarrow c \rightarrow d \), \( d \rightarrow c \rightarrow a \), and \( a \rightarrow b \) respectively. Node b will receive \( \text{msg}(b) = (b, I, d, B, d, I, a, A) \) from node a, where \( \text{msg}(b).p_4 = A \neq \emptyset \). Node b will simply finish its own traversal (Algorithm 3, lines 38-39).

In the above four cases, a type 1 node \( v \) receives the message \( \text{msg}(v) \) initialized by itself, and the message will contain all the information about the traversal. Node \( v \) can then detect different types of change based on the information in \( \text{msg}(v) \) (Algorithm 3, lines 31, 33, 37, and 39). The details about change detection using the message \( \text{msg}(v) \) will be discussed in Algorithm 7.

**Terminal Non-active Nodes of Transition Edges**

If a non-active node \( v \) is the terminal node of a transition edge, then it is a type 2 node. An example of type 2 node can be found in Figure 5.16(b), in which node \( a \) is a type 2 node, since node \( a \) is not an active node and it has a transition edge \((e, a)\). Another example is in Figure 5.17(b), where node \( k \) and \( b \) are both type 2 nodes.

A type 2 node does not need to initialize traversals. The major responsibility of a type 2 node is to receive a message, update the message, and pass the message to the next node on the traversal. A node \( v \) can classify itself as a type 2 node based on Algorithm 4, lines 2-3.

A type 2 node can always receive messages initialized by type 1 nodes (Algorithm 4, line 4). Firstly, if a type 2 node \( v \) receives a message \( \text{msg}(v') \), in which \( \text{msg}(v').p_1 \neq \emptyset \) and \( \text{msg}(v').p_2 = \emptyset \), then node \( v \) is the second KEY node of a traversal (Algorithm 4, lines 5-9). In Figure 5.16(b), for example, node
Algorithm 4: Terminal Non-active Nodes of Transition Edges

Variables: node \( v \); current time step \( t_i \); previous time step \( t_{i-1} \);

if \( v \notin \text{active}(t_i) \) and \( \exists (a, b) \in \text{transitionEdge}(v, t_i), b = v \) then

\( v \) is set as type 2 node at \( t_i \);

if \( v \) is a type 2 node at \( t_i \) and receive \( \text{msg}(v') \) from \( v'' \) then

if \( \text{msg}(v').p_1 \neq \emptyset \) and \( \text{msg}(v').p_2 = \emptyset \) then

\( v \) find the initial node \( v_{\text{init}} \) on the deleted/inserted edge using the clockwise rule;

\( \text{msg}(v').n_2 \leftarrow v; \)

\( \text{msg}(v').p_2 \leftarrow \text{id}_j(v_{\text{init}}, v, t_i - 1); \)

\( v \) send \( \text{msg}(v') \) to \( v_{\text{init}} \) in reverse direction;

else if \( \text{msg}(v').p_3 \neq \emptyset \) and \( \text{msg}(v').p_4 = \emptyset \) then

\( v \) find the initial node \( v_{\text{init}} \) on the deleted/inserted edge using the clockwise rule;

\( \text{msg}(v').n_4 \leftarrow v; \)

\( \text{msg}(v').p_4 \leftarrow \text{id}_j(v_{\text{init}}, v, t_i - 1); \)

\( v \) send \( \text{msg}(v') \) to \( v_{\text{init}} \) in reverse direction;

\( a \) is the second KEY node of the traversal initialized by node \( c \), and node \( a \) should receive \( \text{msg}(c) \) from \( e \). In this case, node \( a \) should update \( \text{msg}(c).n_2 \) and \( \text{msg}(c).p_2 \), and pass the message to the next node on the traversal (Algorithm 4, lines 7-9).

Secondly, if a type 2 node \( v \) receive \( \text{msg}(v') \) with \( \text{msg}(v').p_3 \neq \emptyset \) and \( \text{msg}(v').p_4 = \emptyset \), then node \( v \) is the fourth KEY node of the traversal initialized by \( v' \) (Algorithm 4, lines 10-14). For example, in Figure 5.17(b), the type 1 node \( c \) will start a traversal with \( \text{msg}(c) \), and the type 2 node \( b \) will be the fourth KEY node on the traversal. Node \( b \) should update \( \text{msg}(c).n_4 \) and \( \text{msg}(c).p_4 \) and forward the message to the next node (Algorithm 4, lines 12-14).

In the two examples above, i.e., Figure 5.16(b) and Figure 5.17(b), the type 2 node is the starting node of the second or fourth traversal segments. As already discussed in Section 5.4, the second and fourth segments of a traversal are always in the reverse directions of the directed edges. For example, the second traversal segment, in Figure 5.16(b), is \( a \rightarrow v \rightarrow c \), while the directed edges are \( (v, a) \) and \( (c, v) \). In this case, the type 2 node should find the initial node on deleted/inserted edges as the next node of the traversal (Algorithm 4, line 6 and line 11). For example, in Figure 5.16(b), \( \text{deletedEdge}(a, t_2) = \{(v, a)\} \), so that node \( a \) should traverse to node \( v \), i.e., \( a \rightarrow v \), in the reverse direction of the deleted edge \( (v, a) \).

In some situations, it is possible that a node \( v \) is both a type 1 node and a type 2 node. For example, in Figure 5.15(b), node \( d \) is a non-active node and \( c \) is an active node. Thus, node \( d \) has two transition edges \( (c, d) \) and \( (d, c) \). In
Figure 5.18: An areal object appears between $t_1$ and $t_2$. There are no transition edges, and thus there are no type 1 node and type 2 nodes.

In this case, node $d$ is the initial node of $(d, c)$ and the terminal node of $(c, d)$, i.e., node $d$ is both a type 1 node and a type 2 node. Also, as in Figure 5.15(b), there is an absent segment at node $d$. The absent segments are already included in Algorithm 3, and thus no further actions are required in Algorithm 4.

### Nodes on Deleted/Inserted Edges

A type 3 node can be determined in Algorithm 5, lines 2-3. In Figure 5.16(b), for example, nodes $d$, $e$, and $v$ are type 3 nodes, since node $d$ and $e$ have inserted edges and node $v$ has deleted edges.

A type 3 node does not need to update messages. A type 3 node has two tasks. The first task of a type 3 node is to receive a message initialized from a type 1 node, and simply forwards the message to the next node on the traversal (Algorithm 5, lines 4-12). For example, in Figure 5.16(b), node $d$ would receive a message $msg(c)$ from $c$, and node $d$ would then forward $msg(c)$ to node $e$ based on the clockwise rule.

As already discussed, the second and fourth segments of a traversal are in reverse directions. Thus the type 3 nodes on the second and fourth segments should always forward the message to the initial node of deleted/inserted edges (Algorithm 5, lines 9-12). In Figure 5.16(b), because the type 3 node $v$ is on the second segments of the traversal, node $v$ should send $msg(c)$ to the initial node, i.e., node $c$, of the deleted edges $(c, v)$. In comparison, if a type 3 node $v$ is on the first and third segments of a traversal, then node $v$ should send the message to the terminal node of deleted/inserted edges (Algorithm 5, lines 5-8). In the same example in Figure 5.16(b), node $e$ is on the first segment, so that node $e$ should send $msg(c)$ to the terminal node $a$ of the inserted edge $(e, a)$.

The second task of the type 3 node is independent to the first task. In the case of appearance or disappearance of an areal object, all the nodes in the areal object are active nodes, and thus there is no transition edge. For example,
Algorithm 5: Nodes on Deleted/Inserted Edges

1 Variables: node \( v \); current time step \( t \); previous time step \( t_{i-1} \);
2 if \( v \) is neither a type 1 node nor a type 2 node and \( insertedEdge(v, t_i) \cup deletedEdge(v, t_i) \neq \emptyset \) then
3 \( v \) is set as type 3 node at \( t_i \);
4 if \( v \) is a type 3 node at \( t_i \) and receive \( msg(v') \) from \( v'' \) then
5 if \( (msg(v'), p_1 \neq \emptyset \) and \( msg(v'), p_2 = \emptyset \) or \( (msg(v'), p_3 \neq \emptyset \) and \( msg(v'), p_4 = \emptyset \) then
6 \( v \) find the terminal node \( v_{term} \) on the deleted/inserted edge using the clockwise rule;
7 \( v \) send \( msg(v') \) to \( v_{term} \);
8 else if \( (msg(v'), p_2 \neq \emptyset \) and \( msg(v'), p_3 = \emptyset \) or \( (msg(v'), p_4 \neq \emptyset \) then
9 \( v \) find the initial node \( v_{init} \) on the deleted/inserted edge using the clockwise rule;
10 \( v \) send \( msg(v') \) to \( v_{init} \) in reverse direction;
11 if \( v \) is a type 3 node and receives the traversal started from itself using Algorithm 6 then
12 \( v \) conclude the traversal with no further update of the message \( msg(v^3) \);

In Figure 5.18, an areal object appears between \( t_1 \) and \( t_2 \), and there are six active nodes \( a, b, c, d, e, \) and \( v \). Since there are no transition edges in Figure 5.18, there are no type 1 or type 2 nodes. Nodes \( a, b, c, d, \) and \( e \) are type 3 nodes. Thus, there is no traversal initialized by the traversal approach 1. The five inserted edges \((a, b), (b, c), (c, d), (d, e), \) and \((e, a)\) will remain unvisited.

In this circumstance, Traversal Approach 2 is introduced. In Traversal Approach 2, the initialization of traversals is not based on transition edges. Thus, the second task for the type 3 nodes is to initialize traversals using Traversal Approach 2.

5.5.3 Traversal Approach 2

There is no transition edge in the case of appearance or disappearance, and thus Traversal Approach 1 cannot be used. The objective of Traversal Approach 2 is to traverse all inserted/deleted edges of an areal object when the areal object appears or disappears. For example, in Figure 5.19(b), Traversal Approach 2 should make sure all the inserted edges, i.e., \( \{(a, b), (b, v), (v, a)\} \), of the areal object are visited by a closed traversal.

Firstly, the starting nodes for traversing need to be determined. The set \( eligible(t) \) is used to represent all eligible nodes for starting a traversal. In this approach, all the type 3 nodes are eligible for initializing traversals. In this
case, in Figure 5.19(b), $eligible(t_2) = \{a, b, v\}$. Each node $v$ has a privilege value, denoted as $privilege(v)$. Each eligible node will compare its privilege value with its neighbors to determine the starting nodes. The starting nodes then initialize a traversal to ensure that all inserted/deleted edges of the areal object are visited.

The distance between node $v$ and the sink, denoted as $dist(v, sink)$, is used as an option of privilege values. The shorter distance between node $v$ and the sink, the higher privilege the node $v$ would have. Note that, in Figure 5.19, the sink is located at the bottom left corner of the network, and it is assumed that each node in the network knows the location of the sink. In Figure 5.19(b), node $b$ can be locally selected as a starting node with no communication cost, since $dist(b, sink) < dist(v, sink)$ and $dist(b, sink) < dist(a, sink)$. Similarly, the nodes $a$ and $v$ will not be selected based on the neighborhood comparisons. As soon as the starting node $b$ is determined, the traversals can start. The starting node $b$ will initialize a message, i.e., $msg(b)$, and the message will be passed from one node to the next along the traversal. The closed traversable trail would be $b \rightarrow v \rightarrow a \rightarrow b$. When the traversal returns to its starting node, the traversal will be terminated.

It is possible that multiple starting nodes are selected locally, and thus multiple traversals are initialized at the same time step. For example, in Figure 5.19(d), $eligible(t_2) = \{a, c, d, e\}$. The node $a$ has one neighboring elig-
ble node $e$ and $\text{dist}(a, \text{sink}) < \text{dist}(e, \text{sink})$, so that $a$ will select itself as a starting node. Similarly, $c$ will be another starting node. In this case, potentially there would be two closed trails: $a \rightarrow e \rightarrow d \rightarrow c \rightarrow d \rightarrow e \rightarrow a$ and $c \rightarrow d \rightarrow e \rightarrow a \rightarrow e \rightarrow d \rightarrow c$.

This algorithm will terminate redundant traversals such that only one closed trail remains (Algorithm 6, lines 11-12). Following the example above, the traversal started at $c$ will traverse $c \rightarrow d \rightarrow e \rightarrow a$ with a message $\text{msg}(c)$ and reach another starting node $a$. Suppose the termination of redundant traversals is also based on the rule of shortest distance to the sink, then node $a$ will retrieve the coordinates of node $c$ from $\text{msg}(c)$ for distance calculation. Since $\text{dist}(a, \text{sink}) < \text{dist}(c, \text{sink})$, $a$ will then terminate the traversal from $c$. On the other hand, the traversal started at $a$ will be checked at $c$. Because $a$ has a shorter distance to the sink than $c$, $c$ will allow the traversal from $a$ to continue, and the traversal will return to the starting node $a$: $a \rightarrow e \rightarrow d \rightarrow c \rightarrow d \rightarrow e \rightarrow a$. Only the node which is closest to the sink can receive its own completed traversal.

### 5.5.4 Detecting Change

As discussed in the beginning of Section 5.5, if a node $v$ receives a message $\text{msg}(v)$ initialized by itself, then node $v$ is able to detect different types of changes based on the message $\text{msg}(v)$ (Algorithm 7, line 2).

If the message contains only one segment, i.e., $\text{msg}(v).p_1 \neq \emptyset$ and $\text{msg}(v).p_2 = \emptyset$, then node $v$ detects an appearance or a disappearance (Algorithm 7, lines 3-7). If the segment is on inserted edges, i.e., $\text{msg}(v).p_1 = I$, then node $v$ detects an appearance (Algorithm 7, lines 4-5). If the segment is on deleted edges, i.e., $\text{msg}(v).p_1 \neq I$, then node $v$ detects a disappearance (Algorithm 7, lines 6-7).

If the message contains two segments, i.e., $\text{msg}(v).p_2 \neq \emptyset$ and $\text{msg}(v).p_3 = \emptyset$, then node $v$ would detect an expansion or contraction (Algorithm 7, lines 8-12). As illustrated in Figure 5.13(c)-(d), the first segment of expansion is on inserted edges (Algorithm 7, lines 9-10), while the first segment of contraction is on deleted edges (Algorithm 7, lines 11-12).

There are four types of changes, i.e., merge, split, self-merge, and partial-split, which have four traversal segments (Algorithm 7, lines 13-23). In the cases of merge and self-merge, the first segment is on inserted edges (Algorithm 7, lines 14-18). Merge and self-merge can then be distinguished by the second and fourth segments of traversals. The second and fourth segments have different trajectory identities for a merge, and have the same trajectory identity for a self-merge. While for split and partial-split, the first segment is on deleted edges (Algorithm 7, lines 19-23). The first and third segments have the same trajectory...
Algorithm 6: Traversal Approach 2

Variables: node \( v \); current time step \( t_i \); previous time step \( t_{i-1} \);
1. \( \text{eligible}(t_i) \) represents the set of type 3 nodes, where \( \text{eligible}(t_i) \subseteq V \).

2. if \( v \) is neither a type 1 node nor a type 2 node and \( \text{insertedEdge}(v, t_i) \cup \text{deletedEdge}(v, t_i) \neq \emptyset \) then
3. \( v \) is set as type 3 node at \( t_i \);
4. if \( v \) is a type 3 node, and \( \forall v' \in \text{neighbor}(v) \cap \text{eligible}(t_i) \), \( \text{privilege}(v) > \text{privilege}(v') \) then
5. \( v \) is set as traversal starting node;
6. \( v \) finds the terminal node \( v_{\text{term}} \) on the inserted/deleted edge;
7. \( v \) creates \( msg(v) \);
8. \( msg(v).n_1 \leftarrow v \);
9. \( msg(v).p_1 \leftarrow \text{id}_j(v, v_{\text{term}}, t_{i-1}) \);
10. \( v \) sends \( msg(v) \) to \( v_{\text{term}} \);
11. if \( v' \) is a type 3 node and \( v' \) receives a message \( msg(v) \) then
12. if \( v' \neq v \), \( v' \) is a starting node, and \( \text{privilege}(v') > \text{privilege}(v) \) then
13. the traversal started from \( v \) is terminated at \( v' \);
14. else if \( v' \neq v \), \( v' \) is a starting node, and \( \text{privilege}(v') < \text{privilege}(v) \) then
15. \( v' \) traverses to the next node along inserted/deleted edges using the clockwise rule;
16. else if \( v' \neq v \) and \( v' \) is not a starting node then
17. \( v' \) traverses to the next node along inserted/deleted edges using the clockwise rule;
18. else if \( v = v' \) then
19. the traversal is closed at \( v \); \( v \) detects change using Algorithm 7;

identity for a split, and have different trajectory identities for a partial-split.

5.5.5 Summary of the DETECT Component

The DETECT component consists of two traversal approaches, i.e., Traversal Approach 1 in Section 5.5.2 and Traversal Approach 2 in Section 5.5.3. Traversal Approach 1 requires three types of sensor nodes: type 1, type 2, and type 3 nodes. Traversal Approach 1 can detect expansion, contraction, merge, split, self-merge, and partial-split. In the cases of appearance and disappearance, there are only type 3 nodes. Thus, Traversal Approach 2 is especially designed for type 3 nodes to detect appearance and disappearance.

In Traversal Approach 1, the major responsibility of a type 1 node is to initialize and conclude traversals. And the major responsibility of a type 2 node or a type 3 nodes is to find the next node of a traversal and forward the message to the next node of the traversal.

As introduced in Section 5.4, the traversals form closed trails, therefore the
Algorithm 7: Detecting Change

1 Variables: node $v$;
2 if $v$ receives a message $msg(v)$ initialized by itself then
3    if $msg(v).p_1 \neq \emptyset$ and $msg(v).p_2 = \emptyset$ then
4        if $msg(v).p_1 = I$ then
5            $v$ detects an appearance;
6        else if $msg(v).p_1 \neq I$ then
7            $v$ detects a disappearance;
8    else if $msg(v).p_2 \neq \emptyset$ and $msg(v).p_3 = \emptyset$ then
9        if $msg(v).p_1 = I$ then
10           $v$ detects an expansion;
11        else if $msg(v).p_1 \neq I$ then
12           $v$ detects a contraction;
13    else if $msg(v).p_4 \neq \emptyset$ then
14        if $msg(v).p_1 = I$ then
15           if $msg(v).p_2 \neq msg(v).p_4$ then
16              $v$ detects a merge;
17           else if $msg(v).p_2 = msg(v).p_4$ then
18              $v$ detects a self-merge;
19        else if $msg(v).p_1 \neq I$ then
20           if $msg(v).p_1 \neq msg(v).p_3$ then
21              $v$ detects a partial-split;
22           else if $msg(v).p_1 = msg(v).p_3$ then
23              $v$ detects a split;

active nodes located inside the closed trails do not require message transmission. For example, in Figure 5.9(c), there is a closed trail: $b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow j \rightarrow b$, and there are seven active nodes. Among the seven active nodes, only five active nodes, i.e., $c, d, e, f$, and $g$ have sent messages, while the other two active nodes are located inside the closed trail. The message efficiency of Traversal Approach 1 will be significantly improved when a large number of active nodes are located inside the closed trail. Given a set of active nodes $active(t)$ and $n = |active(t)|$, the complexity of Traversal Approach 1 is $O(\frac{n}{msg\_count})$ [30], since the traversal only visit the boundary, i.e., inserted and deleted edges, of the set of active nodes $active(t)$. An example of a closed trail for a larger set of active nodes can be found in Figure 7.7.

Note that, in Traversal Approach 1, both active nodes and non-active nodes are involved in message transmission. As an example in Figure 5.9(c), the closed trail consists of three non-active nodes, i.e., $b, h$, and $j$. As a result, the total number of messages can be larger than the total number of active nodes, especially when there are a small number of active nodes. For example,
in Figure 5.9(c), there are eight messages during the traversal, but there are only seven active nodes.

In Traversal Approach 1, the closed trails consist of type 1, 2, and 3 nodes, and only type 1 nodes are responsible to initialize traversals. In comparison, in Traversal Approach 2, the closed trails consist of only type 3 nodes, and all of the type 3 nodes are responsible to initialize traversals. All the type 3 nodes will compete with each other, such that only one type 3 node can receive its own traversal. Therefore, the complexity of Traversal Approach 2 is equivalent to the complexity of a leader election algorithm. The most efficient algorithms for leader election in a ring are \(O(m \log m)\), where \(m\) is the number of nodes on the ring [85]. Given \(n = |active(t)|\), the type 3 nodes are only a subset of \(active(t)\), and are located on the boundary, i.e., inserted and deleted edges, of \(active(t)\). Since only type 3 nodes are involved in Traversal Approach 2, the Traversal Approach 2 is linear to the number of active nodes \(O(n)\) [30].

### 5.6 The UPDATE Component

In the UPDATE component, the algorithms for updating trajectory identity have similarities to the algorithms in the DETECT component. Thus, in this section, algorithms for maintaining trajectory identity are not presented in detail again. Instead, Figure 5.20 and 5.21 provide examples to illustrate how trajectory identity can be updated in the sensor network. The UPDATE component has the following characteristics:

- Each inserted edge needs to be provided a trajectory identity, while each deleted edge can simply remove its trajectory identity. For example, Figure 5.20(i) illustrates the stage when trajectory identities have already been updated by the updating process in Figure 5.20(h). In Figure 5.20(i), all the inserted edges, i.e., \((c, d)\), \((d, e)\), and \((e, a)\) have been provided a trajectory identity \(A\). While all the deleted edges, i.e., \((c, v)\) and \((v, a)\), can locally remove their own trajectory identity \(A\).

- Traversals are required for updating trajectory identities. And each traversal is initialized by the node that detects change. For example, in Figure 5.20(h), node \(c\) is a type 1 node, and \(c\) can detect an expansion using Algorithm 7. Thus, node \(c\) should initialize a traversal, i.e., \(c \rightarrow d \rightarrow e \rightarrow a\), to update the trajectory identity. In Figure 5.20(h), the integer numbers in circles shows the edge sequences visited by the traversal.

- In the case of merge, two traversals are required to update trajectory identity. For example, in Figure 5.21(e), a traversal, i.e., \(c \rightarrow b\), is initial-
Figure 5.20: Updating trajectory identities in the case of appearance, disappearance, expansion, and contraction. In (b), (e), (h), and (k), the integer numbers in circles illustrate the traversals for updating trajectory identity. (c), (f), (i), and (l) are the states when trajectory identities have already been updated by the traversals in (b), (e), (h), and (k) respectively.
Figure 5.21: Updating trajectory identities in the case of merge, split, self-merge, and partial-split. In (e) and (h), integer numbers in circles and squares represent two different traversals for updating trajectory identities. (c), (f), (i), and (l), show the updated trajectory identities.
ized by node c to update the trajectory identity A. The other traversal $e \rightarrow k \rightarrow l \rightarrow e$ is initialized by e to update the trajectory identity B.

- In the case of self-merge, each node in a sensor network is not able to locally determine the hole, so that two traversals are required to find the hole of the areal object. For example, in Figure 5.21(h), a region has self-merged into a region with a hole. One traversal, i.e., $a \rightarrow c \rightarrow d \rightarrow e \rightarrow b \rightarrow a$ is initialized by node a, and the other traversal, i.e., $k \rightarrow l \rightarrow m \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow k$ is initialized by node k. The hole can then be determined by the area calculation results, as discussed in Section 4.1.3. The boundary edges of the hole should then be updated appropriately, as shown in Figure 5.21(i).

- The traversals in the UPDATE component only visit the boundaries of areal objects. Thus, given $n = |active(t)|$, the complexity of the UPDATE component is $O\left(\frac{n}{\log n}\right)$.

5.7 Multi-Granularity Sensor Networks

Sections 5.1-5.6 have introduced a decentralized change detection algorithm for a one-level flat planar graph structure. This section introduces a multi-granularity sensor network that has a hierarchical structure. Based on the object-oriented approach, each level of the hierarchy is built as a planar graph. In this case, the planar graph at each level of the hierarchy can inherit the attributes and behaviors of the already developed planar graph in Sections 5.1-5.6.

In particular, this section will focus on the construction of a two-level hierarchical structure with a lower level and an upper level planar graph, because other levels of the hierarchy can be constructed in the same manner recursively, as illustrated in Figure 5.22. The lower level planar graph is already introduced, e.g., in Figure 4.1, and a subset of nodes will be selected to construct the upper level planar graph. Thus, the node density of the upper level will be smaller than the lower level.

As defined in Section 3.3, spatial granularity $\delta_x$ of a WSN is the (average or regular) density of sensors in the deployed area, i.e., $\delta_x = \rho = \frac{|V|}{X}$. Given $G = (V, E)$ as the lower level of the hierarchy, the primary task of this section is to use G to build the upper level planar graph, denoted as $G_p = (V_p, E_p)$, where $V_p \subseteq V$. A built hierarchical network will have two levels of granularities:

$$
\delta_x^l = \frac{|V|}{X}, \quad \delta_x^u = \frac{|V_p|}{X}, \quad \Delta_\delta = \frac{\delta_x^l}{\delta_x^u} = \frac{|V|}{|V_p|}
$$

(5.1)
where $\delta^l_x$ and $\delta^u_x$ represent the granularity of the lower and upper levels respectively, and $\Delta_\delta$ denotes the ratio between two granularities.

Figure 5.23(a) shows an example of a two-level hierarchy structure, where there are 16 nodes in the lower level and three nodes, i.e., a, b, and c, in the upper level. As discussed in Section 4.4.2, the nodes in the second level, i.e., $V_p$, and the nodes in the first level, i.e., $V$, have precedence, or parent-child, relationships. In Figure 5.23(a), for example, node a is the parent of nodes d, e, and g, and node b is the parent of nodes d, e, and f. Notice that the parents a and b are only two-hops away.

The new planar graph $G_p = (V_p, E_p)$ should have a coarser spatial granularity than $G$. Thus, it is defined that $\{p, p'\} \in E_p$ if the two parents can communicate within or less than two hops. Based on the definition, the upper level $G_p$ will consist three edges $\{a, b\}, \{b, c\}, \{c, a\} \in E_p$, as shown in Figure 5.23(b). As emphasized in Section 4.4.2, all the nodes in the hierarchical structure have the same capabilities, e.g., they have the same communication range. The longer edge, e.g., $\{a, b\}$, in the second level represents that $a$ and $b$ have two-hop communication.

88
5.7.1 Dominating Sets

One concept related to the multi-granularity sensor networks is \(k\)-dominating sets. Given a graph \(G = (V, E)\), a \(k\)-dominating set \(S\) requires that each vertex of \(G\) not in \(S\) has at least \(k\) neighbours in \(S\). An example of 2-dominating set is shown in Figure 5.24(b). Given a graph of twelve nodes in Figure 5.24(a), the set of six nodes \(S = \{a, b, c, d, e, f\}\) is a 2-dominating set, since the other six nodes, e.g., node \(g\), have at least two neighbors in the set \(S\).

The concept of 2-dominating set is used to build the multi-granularity sensor network. In a decentralized network, a 2-dominating set can be used as backbones for allocating resources and organizing communication, etc. A minimal 2-dominating set allows for placing the resources or facilities at minimal costs. Also, a 2-dominating set provides an adjacency structure that is valuable for generating topology. The 2-dominating set ensures that for a parent \(p \in V_p\) and a node \(v \in V\), where \(\{p, v\} \in E\), there always exists a parent \(p' \in V_p\) such that \(\{p', v\} \in E\). The objective of the following sections is to select minimal 2-dominating sets in WSNs for building a multi-granularity sensor network. Note that minimal 2-dominating set problem is NP-complete, so that heuristic solutions for the problem are proposed.

5.7.2 Number of Parents

This section examines the theoretical lowest limit of \(V_p\) in a graph \(G\) to ensure that \(V_p\) is a \(k\)-dominating set. \(V_c\) is used to denote the set of children, i.e., \(V_c = \{v : \{v, p\} \in E \text{ and } p \in V_p\}\). It is also defined that if two parents \(p\) and \(p'\) are neighbors, i.e., \(\{p, p'\} \in E\) and \(p, p' \in V_p\), then the two parents are in both sets \(V_p\) and \(V_c\), i.e., \(p, p' \in V_p \cap V_c\). The set of parents of a child \(v\) is denoted by a function \(\text{parent}(v)\), where \(\text{parent}(v) = \{p : \{v, p\} \in E \text{ and }\)
Figure 5.24: An example of a 2-dominating set. In (b), the set of nodes marked by circles is a 2-dominating set.

$p \in V_p \}$ Similarly, child(p) represents all the children of a parent p, where child(p) = {v : {v, p} ∈ E} = neighbor(p).

The following functions are also needed for calculating the minimum number of parents. deg(v) is the degree (the number of edges) of a node v ∈ V. deg(V) is the sum of the degrees of the set V, where deg(V) = ∑ deg(v). And avgdeg(V) is the average degree of the set V, where avgdeg(V) = deg(V)/|V|.

Now suppose each node v ∉ V_p has a fixed number of parents, i.e., |parent(v)| = α (α > 1). Since for a parent p ∈ V_v, child(p) = neighbor(p), the total degrees of all parents is

\[ \text{deg}(V_p) = \sum \text{deg}(p) = \alpha \times |V_c| \] (5.2)

and the estimated number of parents is:

\[ |V_p| = \lceil \frac{\text{deg}(V_p)}{\text{avgdeg}(V)} \rceil = \lceil \frac{\alpha \times |V_c|}{\text{avgdeg}(V)} \rceil \] (5.3)

where \( \lceil x \rceil \) is a ceiling function that gives the smallest integer \( \geq x \). Since ∀v ∉ V_p, |parent(v)| = α and α > 1, thus ∀v ∉ V_p, v ∈ V_c. And as discussed, it is possible that ∃v ∈ V, v ∈ V_p ∩ V_c, so |V_c| ≥ |V| − |V_p|. Hence,

\[ |V_p| \geq \frac{\alpha \times (|V| - |V_c|)}{\text{avgdeg}(V)} \]
\[ |V_p| \times \text{avgdeg}(V) \geq \alpha \times |V| - \alpha \times |V_p| \]
\[ |V_p| \times (\text{avgdeg}(V) + \alpha) \geq \alpha \times |V| \]
\[ |V_p| \geq \frac{\alpha \times |V|}{\text{avgdeg}(V) + \alpha} \]

So, the estimated minimum number of parents is:

\[ |V_p|_{\text{min}} = \lceil \frac{\alpha \times |V|}{\text{avgdeg}(V) + \alpha} \rceil \] (5.4)

Note that Equation 5.4 is an approximation of the minimum number of parents, and the results can be biased by the average degree avgdeg(V) of a
Figure 5.25: Regular Graphs. In the graphs, parents and children are represented as solid circles and empty circles respectively. (a) \( \text{deg}(v) = 2 \), (b) \( \text{deg}(v) = 4 \), and (c) \( \text{deg}(v) = 6 \). The upper level graph \( G_p \) has the same degree as the lower level graph \( G \), as in (d), (e), and (f).

The graphs in Figure 5.25 are subgraph samples of locally finite graphs. In Figure 5.25(a), the graph \( G \) is 2-regular, so that \( \text{avgdeg}(V) = \text{deg}(v) = 2 \). Let \( \alpha = 2 \), then:

\[
|V_p|_{\text{min}} = \left\lceil \frac{\alpha \times |V|}{\text{avgdeg}(V) + \alpha} \right\rceil = \left\lceil \frac{2 \times |V|}{2 + 2} \right\rceil = \left\lceil \frac{|V|}{2} \right\rceil
\]  

(5.5)

Similarly, as illustrated in Figure 5.25(b), \( \text{avgdeg}(V) = \text{deg}(v) = 4 \). If \( \alpha = 4 \) is set to 4, then:

\[
|V_p|_{\text{min}} = \left\lceil \frac{4 \times |V|}{4 + 4} \right\rceil = \left\lceil \frac{|V|}{2} \right\rceil
\]

(5.6)

In Figure 5.25(c), \( \text{avgdeg}(V) = \text{deg}(v) = 6 \). If \( \alpha = 3 \), then:

\[
|V_p|_{\text{min}} = \left\lceil \frac{3 \times |V|}{6 + 3} \right\rceil = \left\lceil \frac{|V|}{3} \right\rceil
\]

(5.7)

The above examples show the calculation of minimal \( k \)-dominating sets in regular graphs. A simple example would be a 2-regular graph with \( |V| = 10 \). If \( k \) is set to 2, then 5 parents can be selected as in Figure 5.25(a) to make sure that each child has exactly 2 parents. And the minimum number of parents is

\[
|V_p|_{\text{min}} = \left\lceil \frac{2 \times 10}{2 + 2} \right\rceil = 5
\]

91
This section has examined the theoretical minimum number of parents, but finding a minimum $k$-dominating set is an NP-complete problem [52]. In this circumstance, heuristic algorithms for the problem are proposed. The proposed algorithms are tasked to find a smallest set of $V_p$ such that the topological requirement, i.e., $\forall v \notin V_p, |parent(v)| > 1$, is satisfied.

In the following two sections, a centralized and a decentralized construction algorithm are proposed. The objective of these two algorithms is to build a new upper communication graph $G_p$, and at the same time minimize the number of nodes in $V_p$.

5.7.3 Centralized Construction Algorithm

The centralized construction algorithm starts with a random selection of the first parent $p_1$ (Algorithm 8, lines 2-3). The set of parents $V_p$ is initialized as empty set. Thus when $p_1$ is selected, $V_p = \{p_1\}$ (Algorithm 8, line 2) and the neighbors of $p_1$ have one parent (Algorithm 8, line 3). Then the second parent $p_2$ is selected such that $p_2$ has the maximum number of common children with $p_1$ compared to any other node in $V$. The set $\text{intersection}(p, p') = \text{child}(p) \cap \text{child}(p')$ is used to denote the common children of two parents $p$ and $p'$.

To avoid two parents next to each other, i.e., $\{p_1, p_2\} \in E$, the second parent $p_2$ should not be the neighbor of the first parent $p_1$. For example, in Figure 5.26(a), the first parent $p_1$ is randomly selected, and $p_2$ is selected as in Figure 5.26(b), such that $p_1$ and $p_2$ are not neighbors in $G$. In Figure 5.26(b), $|\text{intersection}(p_1, p_2)| = 3$, but it is possible that there are more than one parent that have the maximum number of common children with $p_1$, e.g., $|\text{intersection}(p_1, p_2')| = 3$ as in the figure. In this case, the parent that has the shortest distance to the parent $p_1$ will be selected.

The set $\text{child}(V_p)$ is used to represent the set of children of all parents $p \in V_p$, i.e., $\text{child}(V_p) = \bigcup (\text{child}(p) : p \in V_p)$. In Figure 5.26(a), as an example, $V_p = \{p_1\}$ and $\text{child}(V_p) = \text{child}(p_1)$. In Figure 5.26(b), $V_p = \{p_1, p_2\}$ and $\text{child}(V_p) = \text{child}(p_1) \cup \text{child}(p_2)$.

Thus, the third parent $p_3$ should be selected such that $p_3$ has maximum common children with the set $\text{child}(V_p)$, and so on (Algorithm 8, lines 5-11). Figure 5.26(c) shows the fifth selection of the centralized algorithm, and there is a jump from parent $p_4$ to $p_5$. Since the global knowledge of the network structure is available in the centralized algorithm, a new parent, e.g., $p_5$, could be selected at any location away from the previous selection, e.g., $p_4$. Note that Figure 5.26 shows only a subset of $V$ in the graph $G$ as in Figure 5.27(a).

As discussed, the first step of the centralized algorithm is to randomly select one node in the graph as the first parent. In a 500 nodes network as in Figure
Figure 5.26: The first few selections of the centralized construction algorithm. A new parent is selected such that it has the maximum number of common children with the selected parents. In the figure, large dashed-line circles represent the communication range of sensor nodes. Squares are selected parents, solid circles are sensor nodes in the set \( \text{child}(V_p) \), and empty circles are nodes not in the set \( \text{child}(V_p) \).

Figure 5.27: The results of the centralized construction algorithm. (a) One of the results of the first two steps of the algorithm, where some of the nodes \( v \notin V_p \), represented as empty circles, do not satisfy the requirement of \( |\text{parent}(v)| > 1 \) (an example is marked by an arrow) (b) The third step of the algorithm makes sure that all nodes not in \( V_p \) have more than one parent. (c) A subset of the sensor network.

5.27(a), there are potentially 500 different results, if each node in the network is selected as the first parent at each run. All 500 results are computed and a summary is shown in Figure 5.28. Among the 500 results, there are four results having the minimum number of parents of 146. In the worst case, the number of selected parents is 161. The most common results is \( |V_p| = 153 \) with 76 results. All the results approximate a normal distribution. With the best parent selection result \( |V_p| = 146 \), \( \Delta_\delta = \frac{|V|}{|V_p|} = \frac{500}{146} = 3.4 \). In the worst case, \( \Delta_\delta = \frac{500}{161} = 3.1 \), which is still comparable to a quadtree hierarchy. Note that in a quadtree hierarchy each parent node has four children, so that \( \Delta_\delta = 4 \).

One of the intermediate results of the algorithm is shown in Figure 5.27(a),
Algorithm 8: Centralized Construction Algorithm

1. **Step 1:** randomly select the first parent;
2. select any node $v \in V$; $V_p \leftarrow V_p \cup \{v\}; \ child(V_p) = child(v);$
3. for $v' \in neighbor(v)$ do parent($v'$) $\leftarrow$ parents($v'$) $\cup \{v\};$
4. **Step 2:** select new parents that have most common children with the existing parents. But in the decentralized one: a new parent is selected such that the new parent has the maximum number of common children with the existing parents.
5. while $\exists v \in V, v \notin V_p$ and $|\text{parent}(v)| = 0$ do
6. for $v \in V, v \notin V_p$ and $|\text{parent}(v)| = 0$ do
7. intersection($v, V_p) \leftarrow \text{neighbor}(v) \cap \text{child}(V_p);$ 
8. if $\forall v' \in V, |\text{intersection}(v, V_p)| \geq |\text{intersection}(v', V_p)|$ then
9. $V_p \leftarrow V_p \cup \{v\};$
10. child($V_p) \leftarrow \text{child}(V_p) \cup \text{child}(v);$ 
11. for $v' \in \text{neighbor}(v)$ do parent($v'$) $\leftarrow$ parent($v'$) $\cup \{v\};$
12. **Step 3:** select parents for the remaining nodes;
13. while $\exists v \in V, v \notin V_p$ and $|\text{parent}(v)| \leq 1$ do
14. for $v \in V, v \notin V_p$ and $|\text{parent}(v)| \leq 1$ do
15. intersection($v, V_p) \leftarrow \text{neighbor}(v) \cap \text{child}(V_p);$ 
16. if $\forall v' \in V, |\text{intersection}(v, V_p)| \geq |\text{intersection}(v', V_p)|$ then
17. $V_p \leftarrow V_p \cup \{v\};$
18. child($V_p) \leftarrow \text{child}(V_p) \cup \text{child}(v);$ 
19. for $v' \in \text{neighbor}(v)$ do parent($v'$) $\leftarrow$ parent($v'$) $\cup \{v\};$

where $|V_p| = 116$, $\text{deg}(V_p) = 805$, and there are no parents next to each other, i.e. $\forall p \in V_p, |\text{parent}(p)| = 0$. However, Figure 5.27(a) is not the final result. As can be seen in Figure 5.27(a), the empty circles are the non-parent nodes with one parent, noting that most of them are located on the boundary of the sensor network. Some of these non-parent nodes need to be selected as parents to make sure that all the nodes $v \notin V_p$ have more than one parent (Algorithm 8, lines 13-19). For example, the node marked by arrow has only one parent in Figure 5.27(a), so it has been selected as a new parent as in Figure 5.27(b) and (c), where $(p, p') \in E$. Step 3 is the same as Step 2, except that Step 3 allows $(p, p') \in E, p, p' \in V_p$. The final result is shown in Figure 5.27(b), where $|V_p| = 146$, $\text{deg}(V_p) = 972$, and $|\{p : p \in V_p, |\text{parent}(p)| \geq 1\}| = 58$.

### 5.7.4 Decentralized Construction Algorithm

The decentralized construction algorithm uses the same strategies as the centralized one: a new parent is selected such that the new parent has the maximum number of common children with the existing parents. But in the decentralized algorithm, each node needs to make a local decision based on the local knowledge of its neighborhood. In this case any two nodes that are not neighbors can make local decisions without knowing the decisions of each other. In other
words, multiple nodes can be selected as parents by local decisions at different locations simultaneously. The local decisions without the possession of the global knowledge will produce sub-optimal results. However, the simulations in Section 5.7.5 will show that the decentralized sub-optimal results are very close to the centralized results, which implies that the decentralized algorithm provides a good local solution for the parent selection problem. The decentralized algorithm makes the following assumptions about the sensor network.

Assumption 1. Each sensor node in the network has a unique ID.

Assumption 2. The first parent node is pre-selected. For example, the sink can be used as the first parent node. Alternatively, the base station can also send a task message to one of the sensor nodes in the network so that the node received the message can be the first parent node.

Initialization Phase

There are two steps in the initialization phase. In the first step, each sensor node \( v \in V \) broadcasts its unique ID to its neighbors. Each sensor node will then receive and store all the IDs of its neighbors. The set \( \text{neighbor}(v) \) is used to represent the set of neighbors stored in \( v \). In the second step, each node \( v \in V \) broadcasts the set \( \text{neighbor}(v) \). In this case, each node \( v \) will receive a set \( \text{neighbor}(v') \) from each of its neighbors \( v' \in \text{neighbor}(v) \). These neighbors of neighbors, denoted as \( \text{non}(v) \), will be stored in the node \( v \). Formally, \( \text{non}(v) = \{ \text{neighbor}(v') : v' \in \text{neighbor}(v) \} \).

Figure 5.29 illustrates an example of the initialization. The initialization process insures that each node \( v \in V \) in the network contains \( \text{neighbor}(v) \) and \( \text{non}(v) \), which was represented as large solid-line circle and dashed-line circle in Figure 5.29(b) respectively. This local topological information will be used for
Figure 5.29: Initialization phase. (a) The node $v$ received the set of neighbors \textit{neighbor}($v'$) from $v'$ during the initialization (b) The local information stored in the node $v$ after the initialization.

Figure 5.30: Parent selection phase. (a) In step one, parent $p$ broadcasts a cluster formation message to its neighborhood. (b) In step two, each parent candidate $v$, which is two-hop away from the parent $p$, calculate the common children of $p$ and $v$. (c) The parent candidate $v$, who has the maximum number of common children with $p$, become the new parent in step three.

local decisions in the parent selection phase.

**Parent Selection Phase**

In the parent selection phase, a parent $p \in V_p$ is regarded as a cluster head, and the parent and its children \textit{child}($p$) is regarded as a cluster. Thus, the parent selection process is also a process of cluster formation. Three steps are included in this phase, and these three steps need to be repeated until there are no more clusters for formation.

In the first step, eligible parents will broadcast to construct clusters. The first eligible parent is the pre-selected parent in the Assumption 2. Note that each parent is only eligible to broadcast once (Algorithm 9, lines 3-4). As shown in Figure 5.30(a), the initial parent $p$ broadcasts its ID to all its neighbors \textit{neighbor}($p$) (Algorithm 9, lines 3-4). When the neighboring node, e.g., $v_1 \in \textit{neighbor}($p$), receives the message from $p$, $v_1$ will add $p$ to \textit{parent}($v_1$) and
forward the parent’s ID to its neighbors, e.g., $v_5$, which are not in the neighborhood of $p$ (Algorithm 9, lines 5-6). When node $v_5$ receives the ID of $p$ from $v_1$, $v_5$ will become a parent candidate, and store $p$ into a twohopparent($v_5$) set (Algorithm 9, lines 7-8). A two-hop parent of the node $v_5$ is a parent $p \in V_p$ two hops away from the node $v_5$.

In the second step, any node, e.g., $v_5$, that becomes a parent candidate will calculate the common children between $v_5$ and the existing parent $p \in$ twohopparent($v_5$) using the local data set non($v_5$) (Algorithm 9, lines 10-13). In Figure 5.30(b), the parent candidates, e.g., $v_4$, $v_5$, $v_6$, and $v_7$, are represented as empty squares, and the common children of $v_5$ and $p$ are $\{v_1, v_2, v_3\}$. Thus, $\text{intersection}(v_5, p) = \{v_1, v_2, v_3\}$. Similarly, $\text{intersection}(v_5, p) = \{v_1\}$. The time complexity for calculating common children is $O(n^2)$ in each node, and $n$ is a scale to the number of neighboring nodes. Thus, the computation is efficient.

After a parent candidate, e.g., $v_6$, calculates $\text{intersection}(v_6, p)$, $v_6$ will broadcast an integer number $|\text{intersection}(v_6, p)|$ to all its neighbors (Algorithm 9, line 13). Figure 5.30(c) shows an example of local communication of parent candidates in the neighborhood of $v_6$. In the figure, $v_6$ broadcasts $|\text{intersection}(v_6, p)|$ to $\text{neighbor}(v_6)$, and $v_6$ will receive $|\text{intersection}(v_5, p)|$ for example from its neighbor $v_5 \in \text{neighbor}(v_6)$. Then, the node $v_6$ needs to com-

---

### Algorithm 9: Decentralized Construction Algorithm

1. **Local variables:** $isParent(v)$, $isBroadcast(v)$, $isCandidate(v)$
2. **Step 1:** If $v \in V$, $isParent(v)$ is true, and $isBroadcast(v)$ is true then
   - $isBroadcast(v) \leftarrow$ false; broadcast ID($v$) to $\text{neighbor}(v)$
3. If $v \in V$, receive ID($v'$), and $v' \in \text{neighbor}(v)$ then
   - $p$ isBroadcast($v$) $\leftarrow$ parent($v$) $\cup$ \{ $v'$ \}; send ID($v'$) to $\text{neighbor}(v)$ - $\text{neighbor}(v')$
4. If $v \in V$, receive ID($v'$), and $v' \notin \text{neighbor}(v)$ then
   - $isCandidate(v) \leftarrow$ true; twohopparent($v$) $\leftarrow$ twohopparent($v$) $\cup$ \{ $v'$ \}
5. **Step 2:** parent candidates calculate common children;
6. If $v \in V$ and $isCandidate(v)$ is true then
   - $isEligible(v)$ $\leftarrow$ true;
7. intersection($v$) $\leftarrow$ $\text{neighbor}(v)$ $\cap$ child(twohopparent($v$));
8. Broadcast $|\text{intersection}(v)|$ to $\text{neighbor}(v)$;
9. **Step 3:** select new parents;
10. If $v \in V$ and $isCandidate(v)$ is true, and receive $|\text{intersection}(v')|$ then
11. If $|\text{intersection}(v')| \leq |\text{intersection}(v)|$ then $isEligible(v)$ $\leftarrow$ false;
12. If $v \in V$, $isCandidate(v)$ is true and $isEligible(v)$ is true then
13. $isParent(v)$ $\leftarrow$ true;
pare |intersection(v₆, p)| and |intersection(v₅, p)|. If |intersection(v₆, p)| is the maximum integer among the neighbors, v₆ will then become a new parent and is eligible to broadcast cluster formation message (Algorithm 9, line 15-18). In the example in Figure 5.30(c), |intersection(v₆, p)| = 3 is the maximum integer in the neighbors, so v₆ becomes a new parent, while other neighbors keep unchanged.

After the three steps, the first loop of the decentralized cluster algorithm will finish, and the selected parents will start a new loop from Step 1. Note that usually several parent candidates will be selected as parents locally in Step 3, but Figure 5.30(c) only shows one candidate for clarity. The above algorithm is decentralized because it runs in parallel on all sensor nodes, and each sensor node can only access local information about its immediate neighborhood [84].

5.7.5 Simulation Results

The resulting hierarchical structure of the decentralized construction algorithm is similar to its centralized counterpart as in Figure 5.27. Since there is an assumption that the first parent is pre-selected, each of the 500 nodes in the network is pre-selected as the starting parent, and the decentralized construction algorithm is run 500 times to collect 500 results. Among the results, the best results only contains 146 parents, i.e., |p| = 146, and deg(P) = 929. The result |P| = 146 is equal to the best result in the centralized construction algorithm.

Figure 5.31 shows an overall comparison of the 500 results between decentralized and centralized construction algorithms. The figure shows that the decentralized construction algorithm performs similarly to the centralized one. As in Figure 5.31, there are two decentralized results having the minimum number of 146 parents, compared to four centralized results. In the worst case, the decentralized construction algorithm selects 163 parents, while the centralized one selects 161 parents. Generally, the centralized construction algorithm is slightly better than the decentralized one.

Figure 5.32(a) shows the total message transmission in the whole network for a single simulation run. Since the sensor network has 500 sensor nodes, each node has a very small transmission cost. On average, each node only sends about two messages in the parent selection phase of the decentralized construction algorithm. Note that Figure 5.32 does not include the message cost in the initialization phase. In the initialization, each node sends exactly two messages. The above resulting data shows that the decentralized construction algorithm is highly message efficient.

Finally, the algorithm is tested by different network sizes |V| ranging from 200 to 800 nodes. It can be seen from Figure 5.32(b) that the number of messages
Figure 5.31: The results of the decentralized and centralized construction algorithms. The performance of the decentralized algorithm is close to the centralized one, but the centralized algorithm is slightly better.

Figure 5.32: (a) Total message transmitted of a single simulation run in the decentralized construction algorithm. (b) The impact of network size $|V|$ on the communication cost per node.

transmitted by a sensor node is almost unchanged when $|V|$ increases from 200 to 800. Figure 5.32(b) shows that the decentralized construction algorithm is scalable in terms of communication.

5.7.6 Hierarchical Change Detection Algorithm

Sections 5.7.1-5.7.5 have described the automatic establishment of the multi-granularity sensor networks, which consists of multiple levels of planar graphs. The built multi-granularity sensor networks enable change detection in a hierarchical structure.

In Sections 5.1-5.6, the decentralized change detection algorithm has been presented, and the algorithm can only be applied in a one-level flat planar graph structure. Since the multi-granularity sensor network is built using object-oriented approach, the change detection algorithm can be extended to a hierarchical change detection algorithm for the multi-granularity sensor networks. The main characteristics of the hierarchical change detection algorithm are:
Figure 5.33: Hierarchical structure. (a) A planar graph structure $G$ is the lower level of the hierarchy. (b) The upper level of the hierarchy $G_p$. $G_p$ has smaller node density than $G$. (c) The topological relations among parents (squares) and children (circles) is defined by the unit disk (dashed-line circles). (d) An example of multi-hops communication between two parents. (e) Each parent $p$ has a non-overlap polygon to estimate the area of its neighborhood. (f) Area aggregation by merging polygons.

- The lower level of the hierarchical structure is modeled as a directed planar graph $G = (V, E)$, as an example in Figure 5.33(a). The upper level of the hierarchy is also a directed planar graph $G_p = (V_p, E_p)$, as in Figure 5.33(b). The set of parents $V_p$, $V_p \subseteq V$, is selected by parent selection algorithms, as illustrated in Figure 5.33(c). The set of parents of a node $v$ is denoted by $\text{parent}(v)$, where $\text{parent}(v) = \{v' : \{v, v'\} \in E \text{ and } v' \in V_p\}$.

- All the nodes in the hierarchical structure are homogeneous and have the same communication range. For example, in Figure 5.33(d), nodes $h$ and $i$ are neighbors in the second level, but they need to communicate via the path $(h, a, i)$ or $(h, b, i)$. Redundant communication paths usually exist between two parents, e.g., $h$ and $i$, for fault tolerance and routing flexibility.

- In Figure 5.33(e), each parent $p \in V_p$ has a non-overlap polygon of its neighbors, where $\text{neighbor}(p) = \{v : \{v, p\} \in E\}$. Figure 5.33(f) shows
that an arbitrary number of non-overlap polygons can be merged by local communication among parents.

- At the lower level, observations at lower level are made and reported to parents at the upper level. An edge \((p, p') \in E_p\) is defined as an object edge of \(G_p\), if all the nodes on one of the communication paths between \(p\) and \(p'\) are located in an areal object. For example, in Figure 5.33(d), \((h, i)\), and also \((i, h)\), are object edges of \(G_p\), because all the nodes \(h, a,\) and \(i\) on the communication path \((h, a, i)\) are located in an areal object.

- Because the hierarchy is built based on object-oriented approaches, the graph \(G_p\) is operated in exactly the same way as the graph \(G\). The only difference is the communication between two nodes. In \(G_p\), two neighboring parents \(p, p' \in V_p\) are communicating via multi-hops routing, while in \(G\), two neighboring nodes \(v, v' \in V\) can one-hop communicate with each other.
Chapter 6

Evaluation

In this chapter, the change detection algorithms in the previous chapter were evaluated in a simulation environment. The open source agent-based modeling toolkit Repast [107] was used for simulation. This chapter includes two sections. The first section tests the efficiency and accuracy of the change detection algorithm for quantitative analysis. The second section evaluates the scalability, efficiency, and load balance of the algorithm for detecting qualitative spatial changes.

6.1 Quantitative Analysis

The first experiment tests the efficiency of the change detection algorithms for quantitative analysis. There are totally seven cases in Figure 6.1 and 6.2. Each case consists of two time steps, e.g., $t_0$ and $t_1$. In all of these cases, the simulated environment is initialized at $t_0$ with no areal objects. At $t_1$ an areal object abruptly appears. Both Figure 6.1 and 6.2 only show the snapshots at time step $t_1$ when areal objects have already appeared.

As discussed in Section 5.5.3, Traversal Approach 2 should be used for the appearance of areal objects. Thus, this experiment uses Traversal Approach 2 for traversal organization. The change detection algorithm in Sections 5.1-5.6 is run in a one-level flat sensor network structure. In Section 5.7, the two-level hierarchical sensor network structure is introduced. Thus, the hierarchical change detection algorithm in Section 5.7.6 is also evaluated in this experiment as a comparison to the flat change detection algorithm.

The one-level flat structure and two-level hierarchical structure were built in the Repast environment in two separated simulation runs. An example of a flat structure and a hierarchical structure is shown in Figure 6.1(a) and (b) respectively, where 500 sensor nodes are deployed in a region of $800 \times 800$ square
units, with a communication range of 80 units. Note that only object edges are illustrated in the figure.

As introduced in Chapter 1, energy resources are highly constrained for sensor nodes, and wireless communication will intensively consume the constrained energy in sensor nodes. Therefore, a key performance criterion for sensor network algorithms is the level of communication cost. Another important criterion is the error introduced by data abstraction and aggregation in the network. Thus, the accuracy of the algorithms was evaluated by quantitative areas, and efficiency was investigated in terms of number of messages transmitted by sensor nodes.

In the experiment, the sensor network is tasked to detect regions that appear between two consecutive time steps. Figure 6.1 shows an example of appearance, where the whole environment is initialized at $t_0$, and an areal object of 90000 square units (grey area) appears between time step $t_0$ and $t_1$.

In the experiment, there are three separated simulation runs, where the sensor network is tasked to detect the appearance of the areal object using the centralized algorithm, flat change detection algorithm in Sections 5.1-5.6, and hierarchical change detection algorithm in Section 5.7.6 respectively. Figure 6.1 (a) and (b) demonstrate how the object edges are activated in one-level flat structure and two-level hierarchical structure respectively. In a centralized sense-and-transmit algorithm, each active node will simply forward its sensor reading to a sink by multi-hop routings, and the collected information is processed centrally in a base station.
Figure 6.2: Quantitative analysis using flat algorithm and hierarchical algorithm. The first and third rows are one-level flat structure. The second and fourth rows are two-level hierarchical structures.

6.1.1 Experiment Results

As in Table 6.1, the result of the Case 3 shows that the hierarchical algorithm is more message efficient than both centralized algorithm and flat algorithm. Table 6.1 also shows the experimental results with different types of areal objects. Among the results, the hierarchical algorithm is consistently the most efficient algorithm with an approximate 1/3 message transmissions of the centralized
Table 6.1: The comparison of the message transmission and area approximation for appearances of areal objects.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Criteria</th>
<th>Centralized</th>
<th>Flat</th>
<th>Flat Percentage</th>
<th>Hierarchy</th>
<th>Hierarchy Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Message</td>
<td>433</td>
<td>206</td>
<td>47.6%</td>
<td>159</td>
<td>36.7%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>40000</td>
<td>37955</td>
<td>-5.1%</td>
<td>36378.5</td>
<td>-9.1%</td>
</tr>
<tr>
<td>Case 2</td>
<td>Message</td>
<td>467</td>
<td>232</td>
<td>49.7%</td>
<td>157</td>
<td>33.6%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>40000</td>
<td>42451.5</td>
<td>6.1%</td>
<td>37263</td>
<td>-6.8%</td>
</tr>
<tr>
<td>Case 3</td>
<td>Message</td>
<td>1042</td>
<td>561</td>
<td>53.8%</td>
<td>400</td>
<td>38.4%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>90000</td>
<td>95522.5</td>
<td>6.1%</td>
<td>85379</td>
<td>-5.1%</td>
</tr>
<tr>
<td>Case 4</td>
<td>Message</td>
<td>1417</td>
<td>704</td>
<td>49.7%</td>
<td>493</td>
<td>34.8%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>120000</td>
<td>132647</td>
<td>10.5%</td>
<td>120808</td>
<td>0.7%</td>
</tr>
<tr>
<td>Case 5</td>
<td>Message</td>
<td>836</td>
<td>388</td>
<td>46.4%</td>
<td>266</td>
<td>31.8%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>70685</td>
<td>73019</td>
<td>3.3%</td>
<td>71366</td>
<td>1.0%</td>
</tr>
<tr>
<td>Case 6</td>
<td>Message</td>
<td>1200</td>
<td>632</td>
<td>52.7%</td>
<td>449</td>
<td>37.4%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>105558</td>
<td>106092</td>
<td>0.5%</td>
<td>100060.5</td>
<td>-5.2%</td>
</tr>
<tr>
<td>Case 7</td>
<td>Message</td>
<td>1208</td>
<td>557</td>
<td>46.1%</td>
<td>429</td>
<td>35.5%</td>
</tr>
<tr>
<td></td>
<td>Area</td>
<td>105558</td>
<td>108463</td>
<td>2.8%</td>
<td>97456</td>
<td>-7.7%</td>
</tr>
</tbody>
</table>

algorithm, while the message costs of the flat algorithm are about 1/2 of the centralized one.

The change detection algorithms are tested with three types of regions, i.e., convex regions (Case 1, 2 and 5), concave regions (Case 3 and 6), and regions with a hole (Case 4 and 7). The results show that both hierarchical algorithm and flat algorithm are less efficient to detect concave regions than convex regions and regions with a hole. The reason is that a traversal of boundary edges needs to visit more edges in concave regions than other types of regions, as illustrated in Figures 6.1 and 6.2. Also, in the concave regions, more nodes are likely to be selected as starting nodes, resulting in more redundant traversals on boundary edges. On the other hand, the results in Table 6.1 show that the regions with a hole in Case 4 and 7 do not reduce the efficiency of both flat and hierarchical algorithms, because a hole in a region can be locally determined by the traversal orientation introduced in Section 4.1.3.

In terms of the area of changed regions, both flat and hierarchical algorithms have fairly good approximation results. Since the hierarchical algorithm aggregates changed regions at the upper level of the hierarchy and thus has a coarser level of detail than the flat algorithm, the results of area calculation in the hierarchical algorithm are less consistent than the results of the flat algorithm, as listed in Table 6.1.

6.1.2 Discussion

The experiment results show that the hierarchical algorithm is more efficient in communication than a centralized algorithm for quantitative analysis. Also, the hierarchical algorithm is more efficient than the flat algorithm for the reason
The decentralized algorithms require traversals of boundary edges to aggregate and calculate changed areas, and the number of nodes for traversal in the one-level flat structure $|V|$ is much higher than the two-level hierarchical structure $|V_p|$. In this case, there will be more object cycles and object edges in the flat structure, as illustrated in Figure 6.1 and 6.2. Consequently, there will be more boundary edges in the one-level flat structure. If there are more boundary edges, then there will likely be more starting nodes for traversals, resulting in more redundant traversals on boundary edges. Thus, the flat structure is less efficient in traversals than the hierarchical structure.

6.2 Detecting Qualitative Changes

The second experiment tests the change detection algorithms for detecting qualitative spatial changes. The change detection algorithm in Sections 5.1-5.6 is applied for this experiment. The scalability, efficiency, and load balancing of the change detection algorithm are evaluated in three sub-experiments.

6.2.1 Experiment 2.1: Scalability

In the experiment for scalability, there are about 2000 simulation runs, and each type of change had about 250 runs. An areal object is simulated as one or several $r \times r$ square boxes, as examples in Figure 6.3. From the 1st to the 2000th simulation runs, the parameter $r$ gradually increase, and thus the number of active nodes should also gradually increase. At each simulation run, a specific type of spatial change can occur at randomized locations within the sensor network. Figure 6.3 (c) and (d), for example, show merge and partial-split that had occurred at different locations of the sensor network.

The change detection algorithm was installed in each sensor node in the network. In all the simulation runs, the eight different types of changes were correctly detected by the sensor network, as long as the granularity of observation is set up appropriately based on the granularity of areal objects.

All three components of the change detection algorithm, i.e., BROADCAST, DETECT, and UPDATE, require message transmissions. In the BROADCAST component, each active node at a given time step $t_i$ should broadcast its status to its neighbors. As introduced in Section 5.1, Algorithm 1 is used in the BROADCAST component to determine boundary edges of areal objects, and Algorithm 2 is used to calculate inserted and deleted edges. Clearly, Algorithm 1 requires one message per active node, and Algorithm 2 does not require message transmission.
After the calculation of inserted and deleted edges in the BROADCAST component, Algorithm 3-7 is used in the DETECT component for change detection. Algorithm 3-6 require message transmissions, while Algorithm 7 does not need messages. In the DETECT component, messages are expended on the traversals of inserted and deleted edges. And as discussed in Section 5.4, all of the traversals for change detection will form closed traversable trails.

The last component of the change detection algorithm is the UPDATE component, which is used to update the trajectory identity of areal objects as in Section 5.6. The UPDATE component requires traversals on boundary edges. In both DETECT and UPDATE components, the initialization and termination of the traversals are well organized, therefore message transmission during the traversals should be efficient.

**Experiment Results**

The total message costs for the combination of BROADCAST and DETECT components are illustrated in Figure 6.4. For example, in the case of expansion in Figure 6.4(c), the total messages sent is around 80 when there are 50 active nodes. The 80 messages include 50 messages for the BROADCAST component and \( 80 - 50 = 30 \) messages for the DETECT components. The total messages costs for the UPDATE component is in Figure 6.5. During different simulation runs, the number of active nodes gradually increases from 1 to about 50, as can be seen in Figure 6.4 and 6.5.

The scalability is measured by number of messages sent with increasing number of active nodes. For the BROADCAST and DETECT components, the
Table 6.2: Summary of the regression analysis result \((y = a \times x^b)\) for the combination of BROADCAST and DETECT components.

<table>
<thead>
<tr>
<th>Types of change</th>
<th>Factor (a)</th>
<th>Power (b)</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>2.2973</td>
<td>0.9415</td>
<td>0.9564</td>
</tr>
<tr>
<td>Disappearance</td>
<td>2.3347</td>
<td>0.9353</td>
<td>0.9551</td>
</tr>
<tr>
<td>Expansion</td>
<td>3.3547</td>
<td>0.8046</td>
<td>0.9930</td>
</tr>
<tr>
<td>Contraction</td>
<td>3.4046</td>
<td>0.8014</td>
<td>0.9926</td>
</tr>
<tr>
<td>Merge</td>
<td>6.4495</td>
<td>0.6785</td>
<td>0.9898</td>
</tr>
<tr>
<td>Split</td>
<td>7.0449</td>
<td>0.6552</td>
<td>0.9877</td>
</tr>
<tr>
<td>Self-merge</td>
<td>6.4553</td>
<td>0.6934</td>
<td>0.9893</td>
</tr>
<tr>
<td>Partial-split</td>
<td>6.2780</td>
<td>0.6975</td>
<td>0.9896</td>
</tr>
</tbody>
</table>

Table 6.3: Summary of the regression analysis result \((y = a \times x^b)\) for the UPDATE component.

<table>
<thead>
<tr>
<th>Types of change</th>
<th>Factor (a)</th>
<th>Power (b)</th>
<th>(R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>1.5363</td>
<td>0.7421</td>
<td>0.9712</td>
</tr>
<tr>
<td>Disappearance</td>
<td>0</td>
<td>0</td>
<td>n/a</td>
</tr>
<tr>
<td>Expansion</td>
<td>2.2226</td>
<td>0.5952</td>
<td>0.9652</td>
</tr>
<tr>
<td>Contraction</td>
<td>0.5706</td>
<td>0.6444</td>
<td>0.8089</td>
</tr>
<tr>
<td>Merge</td>
<td>4.8358</td>
<td>0.5143</td>
<td>0.9535</td>
</tr>
<tr>
<td>Split</td>
<td>1.8487</td>
<td>0.7314</td>
<td>0.9465</td>
</tr>
<tr>
<td>Self-merge</td>
<td>22.443</td>
<td>0.4862</td>
<td>0.9418</td>
</tr>
<tr>
<td>Partial-split</td>
<td>5.2068</td>
<td>0.5217</td>
<td>0.9024</td>
</tr>
</tbody>
</table>

results of power regression analysis for the eight different types of changes is shown in Figure 6.4(a)-(h) respectively. Table 6.2 lists the results of regression analysis for the eight types of changes. A power regression \((y = a \times x^b)\) is used to fit the 250 plotted results for each type of change. All the regression results achieve high goodness of fit, with the range of \(R^2\) from 0.9551 to 0.993. As in Table 6.2, there is \(b < 1\) for all the eight types of changes, and thus the BROADCAST and DETECT components should have an order of \(O(n)\) or less. The experiment shows that the BROADCAST and DETECT components is highly scalable.

Figure 6.5 shows the results for the UPDATE component. The \(X\) axis is the number of active nodes and the \(Y\) axis is total messages sent. Notice that the scale of \(Y\) axis in Figure 6.5(g) is 150, while all the other seven results have the same scale of 50. Self-merge consumes the most messages for updating trajectory identity, because when an areal object self-merges into a region with a hole, there will be two closed trails of boundary edges: one is internal for the hole and the other is external for the areal object. Also, the two closed trails can not be distinguished locally by a single sensor node. To ensure trajectory
identity is properly updated, two additional traversals on the two closed trails is required to determine the hole. In this case, self-merges will spend more messages especially for the determination of the hole.

The power regression analysis results for the UPDATE component are listed in Table 6.3. The regression results for the UPDATE component also have high goodness of fit, where the range of $R^2$ for appearance, expansion, merge, split, self-merge, and partial-split is between 0.9024 and 0.9712. Contraction has $R^2 = 0.8089$, and $R^2$ is not available for disappearance. Similar to the BROADCAST and DETECT components, the UPDATE component also have an order of $O(n)$ or less, since the power $b$ is ranged from 0.4862 to 0.7421, except $b = 0$ for disappearance.

To compare with the decentralized change detection algorithm, the experiment also implements a centralized sense-and-transmit algorithm, in which each active node will simply forward its sensor reading to a sink by multi-hop routings. The power regression results for the centralized algorithm is: $a = 11.6970$, $b = 1.1079$, and $R^2 = 0.9984$. By comparing power $b$ in $y = ax^b$, it is clear that the decentralized algorithm is more scalable than a centralized algorithm.

**Discussion**

The experiment proves that the decentralized change detection algorithm is highly scalable, since all the components of the algorithm are scalable and have an order of $O(n)$ or less. Generally, the power $b$ for the combination of BROADCAST and DETECT components, ranged from 0.6552 to 0.9415, is higher than $b$ for the UPDATE component. The combination of BROADCAST and DETECT components has a stronger correlation ($R^2$) than the UPDATE component, because the traversals for DETECT components always form closed traversal trails. In comparison, the traversals for the UPDATE component do not form closed trails, as illustrated in Figure 5.20 and 5.21. In this case, the message costs for the UPDATE component can vary more significantly than that of the BROADCAST and DETECT components.

For the BROADCAST and DETECT components, the plotted results for appearance and disappearance in Figure 6.4(a)-(b) show less correlations than the other six types of changes in Figure 6.4(c)-(h). The $R^2$ for appearance and disappearance are 0.9564 and 0.9551, while the other six types of changes have $R^2$ ranged from 0.9877 to 0.9930. As introduced in Section 5.5, there are two different traversal approaches for change detection. Traversal Approach 1 is organized by transition edges, while Traversal Approach 2 does not use transition edges.

In the case of appearance or disappearance of an areal object, all the nodes in
Figure 6.4: Total messages sent for the BROADCAST and DETECT components with increasing number of active nodes.
Figure 6.5: Total messages sent for the UPDATE component with increasing number of active nodes.
the areal object are active nodes, and thus there is no transition edge. Traversal Approach 2 is then needed for appearance or disappearance of areal objects, and Algorithm 6 is proposed in Traversal Approach 2 to organized traversals. In Algorithm 6, the starting node of the traversal is determined based on the rule of shortest distance to the sink, therefore it is possible that multiple starting nodes are selected locally and multiple traversals will be initialized. The number of traversals can be different for different simulation runs, so that the number of messages sent during these traversals can also vary correspondingly, resulting in the less correlated results in Figure 6.4(a)-(b).

As discussed in Section 5.6, traversals are not required for updating trajectory identity in the case of disappearance. Thus, there is no message cost in the UPDATE component for disappearance, as shown in Figure 6.5(b). Also, contraction has the weakest correlation with \( R^2 = 0.8089 \) for the UPDATE component, compared to the other types of changes.

Figure 6.6 shows the same result as Figure 6.5(d), but the scale of \( Y \) axis is 10 in Figure 6.6. As in Figure 6.6, the total messages sent for contraction is no more than 10 when the number of active nodes increases to around 50. Also, the numbers of total messages are integers, resulting in weaker correlation.

![Figure 6.6: Total messages sent for the UPDATE component in the case of contraction. The result in this figure is the same as Figure 6.5(d) but with a different scale.](image)

### 6.2.2 Experiment 2.2: Efficiency

In this experiment, the efficiency of the decentralized change detection algorithm is tested. The three components of the algorithm, i.e., the BROADCAST, DETECT, and UPDATE components, are evaluated respectively. The efficiency of the components is evaluated by the number of messages sent per active node. This experiment uses the results in Section 6.2.1, and sample data is selected
Table 6.4: Average messages sent per active node for the DETECT component.

<table>
<thead>
<tr>
<th>Active Nodes</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>1.00</td>
<td>0.79</td>
<td>0.97</td>
<td>0.67</td>
</tr>
<tr>
<td>Disappearance</td>
<td>1.04</td>
<td>0.92</td>
<td>1.05</td>
<td>0.66</td>
</tr>
<tr>
<td>Expansion</td>
<td>1.15</td>
<td>0.86</td>
<td>0.73</td>
<td>0.65</td>
</tr>
<tr>
<td>Contraction</td>
<td>1.18</td>
<td>0.85</td>
<td>0.74</td>
<td>0.65</td>
</tr>
<tr>
<td>Merge</td>
<td>2.06</td>
<td>1.38</td>
<td>1.21</td>
<td>1.03</td>
</tr>
<tr>
<td>Split</td>
<td>2.18</td>
<td>1.40</td>
<td>1.19</td>
<td>1.06</td>
</tr>
<tr>
<td>Self-merge</td>
<td>2.13</td>
<td>1.59</td>
<td>1.31</td>
<td>1.17</td>
</tr>
<tr>
<td>Partial-split</td>
<td>2.07</td>
<td>1.53</td>
<td>1.30</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Table 6.5: Average messages sent per active node for the UPDATE component.

<table>
<thead>
<tr>
<th>Active Nodes</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>0.86</td>
<td>0.73</td>
<td>0.64</td>
<td>0.57</td>
</tr>
<tr>
<td>Disappearance</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Expansion</td>
<td>0.90</td>
<td>0.64</td>
<td>0.58</td>
<td>0.51</td>
</tr>
<tr>
<td>Contraction</td>
<td>0.26</td>
<td>0.22</td>
<td>0.16</td>
<td>0.15</td>
</tr>
<tr>
<td>Merge</td>
<td>1.65</td>
<td>1.10</td>
<td>1.00</td>
<td>0.81</td>
</tr>
<tr>
<td>Split</td>
<td>1.01</td>
<td>0.85</td>
<td>0.77</td>
<td>0.69</td>
</tr>
<tr>
<td>Self-merge</td>
<td>6.84</td>
<td>4.73</td>
<td>4.09</td>
<td>3.78</td>
</tr>
<tr>
<td>Partial-split</td>
<td>1.79</td>
<td>1.21</td>
<td>1.10</td>
<td>0.88</td>
</tr>
</tbody>
</table>

from the 2000 simulation results. For example, there are about 250 simulation results for appearances, and there are eight results when the number of active nodes is 20. In the DETECT component, the result of 0.79 message sent per active node for appearances with 20 active nodes, as in Table 6.4, is the average of the eight simulation runs.

Experiment Results

In the BROADCAST component, each active node always broadcasts one message to its neighbors. Thus, there is always one message sent per active node for the BROADCAST component. Tables 6.4 and 6.5 show the average messages sent per active node for the DETECT and UPDATE components. Active node numbers of 10, 20, 30, and 40 are picked as samples.

In the DETECT component, merge, split, self-merge, and partial-split have close results. They spend around two messages per active node when the number of active nodes is 10, and the message sent per active node decreases to about one when the number of active nodes is 40, as listed in Table 6.4. The four types of changes has close results because the four traversals for detecting these four changes have similar structures, as in Figure 5.13(e)-(h). In each of the four traversals, there are four segments and two initial non-active nodes (type 1 node), as introduced in Section 5.5. Thus, two traversals will always be
initialized: one traversal will traverse four segments and return to its origin node, and the other traversal will traverse two segments and terminate by a type 1 node. If a closed trail is regarded as one traversal unit, then in the cases of merge, split, self-merge, and partial-split, there will be always 1.5 traversal units.

In the cases of expansion and contraction, there are two segments as in Figure 5.13(c) and (d), and there is only one type 1 node, as in Figure 5.9. Therefore, only one traversal will be initialized, and the traversal will be closed by its origin node. Expansion and contraction will have only one closed trail. As can be seen in Table 6.4, the average messages sent per active node for merge, split, self-merge, and partial-split is always about 1.5 times more than expansion and contraction.

For appearance and disappearance, there would be some uncertainty for average messages, because the number of traversals may be different at different simulation runs, as already discussed in Section 6.2.1. But the results in Table 6.4 show that the traversals for appearance and disappearance are efficient. Especially, for example, when the number of active nodes is 10, the average messages for appearance and disappearance (1.00 and 1.04) can be less than the average messages for expansion and contraction (1.15 and 1.18). The reason is that only active nodes are involved in the decentralized change detection algorithm for appearance and disappearance as illustrated in Figure 5.8. While in the cases of expansion and contraction, both active and non-active nodes will be involved in the algorithm. For example, in an expansion in Figure 5.9(a), there are seven active nodes, but ten nodes, including seven active nodes and three non-active nodes, are involved in the algorithm.

In the results of UPDATE component in Table 6.5, self-merge spends the most messages among the eight types of changes, as discussed in the previous section. Also, no message is required in the UPDATE component for disappearance. Among the seven types of changes except disappearance, contraction is most messages efficient, and expansion is more efficient than merge, split, self-merge and partial-split.

Figure 5.20 and 5.21 provide some general ideas about the messages costs in the UPDATE component. In Figure 5.20(h) and (k) and Figure 5.21(b), (e), (h), and (k), all of the six changes have two active nodes. Contraction requires only two messages, and expansion requires three messages. Merge and partial-split, illustrated in Figure 5.21(b) and (k) respectively, have similar traversal structures. And the results for merge and partial-split in Table 6.5 are closest to each other, compared to the other six types of changes. Split requires two traversals in the UPDATE component, but actually split uses less messages than merge and partial-split, as in Table 6.5. Figure 5.21(e) shows an example of the
Table 6.6: Sum of average messages sent per active node for BROADCAST, DETECT and UPDATE components.

<table>
<thead>
<tr>
<th>Active Nodes</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>2.86</td>
<td>2.52</td>
<td>2.61</td>
<td>2.24</td>
</tr>
<tr>
<td>Disappearance</td>
<td>2.04</td>
<td>1.92</td>
<td>2.05</td>
<td>1.66</td>
</tr>
<tr>
<td>Expansion</td>
<td>3.05</td>
<td>2.50</td>
<td>2.30</td>
<td>2.16</td>
</tr>
<tr>
<td>Contraction</td>
<td>2.44</td>
<td>2.07</td>
<td>1.90</td>
<td>1.80</td>
</tr>
<tr>
<td>Merge</td>
<td>4.71</td>
<td>3.48</td>
<td>3.20</td>
<td>2.84</td>
</tr>
<tr>
<td>Split</td>
<td>4.19</td>
<td>3.25</td>
<td>2.96</td>
<td>2.75</td>
</tr>
<tr>
<td>Self-merge</td>
<td>9.96</td>
<td>7.32</td>
<td>6.39</td>
<td>5.95</td>
</tr>
<tr>
<td>Partial-split</td>
<td>4.85</td>
<td>3.74</td>
<td>3.40</td>
<td>3.02</td>
</tr>
</tbody>
</table>

**Discussion**

Generally, Tables 6.4 and 6.5 show the message efficiency is improved when the number of active nodes increases. In both DETECT and UPDATE components, the average message cost per active node reduces about 50% when the number of active nodes increases from 10 to 40. For example, in the case of merge in Tables 6.4, the average messages is 2.06 with 10 active nodes and the average messages decrease to 1.03 with 40 active nodes.

Table 6.6 is simply a sum of the average messages sent per active node for the BROADCAST, DETECT and UPDATE components. For example, in the case of expansion with 10 active nodes, the BROADCAST, DETECT and UPDATE components use 1, 1.15, and 0.9 messages respectively, therefore the total message cost is 3.05, as in Table 6.6.

When the number of active nodes is 10, the total number of message sent are 2-3 for appearance, disappearance, expansion and contraction, 4-5 for merge, split, and partial-split, and about 10 for self-merge. The centralized sense-and-transmit algorithm is again used as the benchmark of the decentralized change detection algorithm. In the centralized algorithm, each active node will simply report to a sink by multi-hop routings. If, for example, there are 10 active nodes, and the average hop distance between the active nodes and the sink is around 4-5 hops, then the cases of appearance, disappearance, expansion and contraction in the decentralized change detection algorithm will be more efficient than the centralized algorithm, the cases of merge, split, and partial-split will be as efficient as the centralized algorithm, and partial-split will be less efficient than the centralized one.

However, when the network size is becoming larger, for example $40 \times 40 = 1600$ nodes, the average hop distance between active nodes and the sink will be larger, and the decentralized change detection algorithm can then well outper-
form the centralized algorithm. Also, if the number of active nodes increases, e.g., from 10 to 40, the decentralized change detection algorithm will also become more efficient, as shown in Table 6.6.

Finally, in Tables 6.4, 6.5, and 6.6, the maximum number of active nodes is 40. When the number of active nodes is more than 40, in some simulation runs, the areal objects may be outside the boundary of the sensor network. Consequently, there will be some unexpected errors that are not caused by the change detection algorithm. The detection of the boundary of a sensor network is not included in the change detection algorithm, and it requires future investigations.

6.2.3 Experiment 2.3: Load Balancing

To test the load balance of the decentralized change detection algorithm, this experiment has 32 sample simulation cases. For each type of change, there are four simulation cases, and the parameter \( r \) has different values in different cases, as listed in Table 6.7. For the four cases 1.1, 1.2, 1.3, and 1.4, for example, the parameter \( r \) is randomly set in the range of 80-100, 120-140, 160-180, and 200-220 units, and the number of active nodes is 11, 14, 22, and 34 respectively. The other cases have similar \( r \) values as in Table 6.7.

Appearance and Disappearance

Figure 6.7(a) illustrates the snapshots for Case 1.1 in Table 6.7, where an areal object appears between the two snapshots and there are eleven active nodes. As discussed in Section 6.2.1, the BROADCAST, DETECT, and UPDATE components require message transmission. For Case 1.1, the details of messages sent at each node for the three components are:

- **BROADCAST**: one message per active node for \( \{a, b, c, d, e, f, g, h, i, j, k\} \).
- **DETECT**: \( a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow (a) \).
- **UPDATE**: \( a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow (a) \).

As illustrated in Figure 6.7, the traversal in the DETECT component is highlighted by solid-line circles, and the traversal in the UPDATE component is marked with slightly larger dashed-line circles. Notice that for the appearance, the starting node for the DETECT component is selected based on the rule of shortest distance to the sink (the sink is located in the bottom left corner of the sensor network). In the Case 1.1, only node \( a \) will be selected as starting node, and thus there is only one traversal in the DETECT component. As a result, nine nodes \( \{a, b, c, d, e, f, g, h, i\} \), which are located on the boundary edges on the areal object, transmit three messages, and two nodes \( \{j, k\} \), which are inside the areal object, transmit only one message, as listed in Case 1.1 in Table 6.7.
Figure 6.7: (a) Snapshots for Case 1.1 (appearance) in Table 6.7. (b) Snapshots for Case 2.1 (disappearance) in Table 6.7. A node with a solid-line circle represents that the node is involved in the traversal of the DETECT component. A node with a slightly larger dashed-line circle represents that the node is involved in the UPDATE traversal.

Note that the last node of a traversal (marked by brackets above) does not need to transmit a message.

Figure 6.7(b) shows an example of disappearance, and the result is listed in Case 2.1 in Table 6.7. The detail messages sent at each node are:

**BROADCAST:** one message per active node for \{a, b, c, d, e, f, g, h, i, j, k\}.

**DETECT:** \(a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow j \rightarrow (a)\).

**UPDATE:** none.

Similar to Case 1.1, only one node \(a\) has been selected as starting node, and there is only one traversal in the DETECT component. In the case of disappearance, no traversal is required in the UPDATE component. Thus, \(\{k\}\) transmits one message, and \(\{a, b, c, d, e, f, g, h, i, j\}\) transmit two messages, as summarized in Case 2.1 in Table 6.7.

As first been introduced in Section 5.5.3, for appearance and disappearance, it is possible that multiple starting nodes are selected locally, and thus multiple traversals are initialized in the DETECT component. And the number of traversals in the DETECT component can be different, depended on different simulation cases. In this case, the load balance, especially for the nodes on the
### Table 6.7: The number of nodes that transmit 1, 2, 3, 4, or 5 messages for the eight different types of changes.

<table>
<thead>
<tr>
<th>Types of change</th>
<th>Cases</th>
<th>r</th>
<th>1 msg</th>
<th>2 msgs</th>
<th>3 msgs</th>
<th>4 msgs</th>
<th>5 msgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>1.1</td>
<td>80-100</td>
<td>2</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>120-140</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>160-180</td>
<td>6</td>
<td></td>
<td>14</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.4</td>
<td>200-220</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disappearance</td>
<td>2.1</td>
<td>80-100</td>
<td>1</td>
<td></td>
<td>10</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>120-140</td>
<td>2</td>
<td></td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2.3</td>
<td>160-180</td>
<td>14</td>
<td></td>
<td>16</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4</td>
<td>200-220</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expansion</td>
<td>3.1</td>
<td>80-100</td>
<td>2</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>120-140</td>
<td>10</td>
<td></td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>160-180</td>
<td>17</td>
<td></td>
<td>1</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.4</td>
<td>200-220</td>
<td>23</td>
<td></td>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Contraction</td>
<td>4.1</td>
<td>80-100</td>
<td>2</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.2</td>
<td>120-140</td>
<td>8</td>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.3</td>
<td>160-180</td>
<td>14</td>
<td></td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.4</td>
<td>200-220</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge</td>
<td>5.1</td>
<td>80-100</td>
<td>2</td>
<td></td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5.2</td>
<td>120-140</td>
<td>16</td>
<td></td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>5.3</td>
<td>160-180</td>
<td>30</td>
<td></td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5.4</td>
<td>200-220</td>
<td>40</td>
<td></td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Split</td>
<td>6.1</td>
<td>80-100</td>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.2</td>
<td>120-140</td>
<td>14</td>
<td></td>
<td>9</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.3</td>
<td>160-180</td>
<td>27</td>
<td></td>
<td>13</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.4</td>
<td>200-220</td>
<td>36</td>
<td></td>
<td>13</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Self-merge</td>
<td>7.1</td>
<td>80-100</td>
<td>50</td>
<td></td>
<td>11</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7.2</td>
<td>120-140</td>
<td>50</td>
<td></td>
<td>13</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>7.3</td>
<td>160-180</td>
<td>64</td>
<td></td>
<td>19</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>7.4</td>
<td>200-220</td>
<td>88</td>
<td></td>
<td>26</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Partial-split</td>
<td>8.1</td>
<td>80-100</td>
<td>13</td>
<td></td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.2</td>
<td>120-140</td>
<td>19</td>
<td></td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.3</td>
<td>160-180</td>
<td>29</td>
<td></td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.4</td>
<td>200-220</td>
<td>40</td>
<td></td>
<td>12</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Boundary edges of areal objects, can vary at different simulation runs. Cases 1.1-1.4 and 2.1-2.4 in Table 6.7 illustrate the variety of load balancing for appearance and disappearance. However, good load balance can still be achieved in the cases of appearance and disappearance, with all nodes transmitting no more than five messages.

### Expansion and Contraction

Compared to appearance and disappearance, expansion and contraction are more consistent in load balance, as can be seen in Table 6.7, Cases 3.1-3.4 and 4.1-4.4. Figure 6.8(a) provides an example of expansion, and the result of the example is listed in Case 3.1 in Table 6.7. The detail message transmissions are listed below:

**Broadcast:** one message per active node for \{e, f, g, h\}.
Figure 6.8: (a) Snapshots for Case 3.1 (expansion) in Table 6.7. The different dashed-line circle (with larger gap) at node a represents that node a only receives message in the UPDATE component. (b) Snapshots for Case 4.1 (contraction) in Table 6.7.

DETECT: \[ b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow a \rightarrow (b). \]

UPDATE: \[ b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow (a). \]

In this example, there are six active nodes for the BROADCAST component. In the DETECT component, there is always one closed traversal for expansion, referred to the discussion in Section 5.4.2. In this example, node b will initialize the traversal for the DETECT component, and the traversal will be closed at the origin node b. Node b should then initialize a traversal for the UPDATE component, and the update traversal will be terminated by node a. In Figure 6.8(a), the different dashed-line circle (with larger gap) at node a represents that node a only receives message in the UPDATE component. The organization of the traversal in the UPDATE component is illustrated in Figure 5.20(h). The nodes that have transmitted one, two, and three messages are \{a, h\}, \{b\}, and \{c, d, e, f, g\} respectively, as in Table 6.7, Case 3.1. Note that there are six active nodes, but eight nodes have transmitted messages, where a and b are not active nodes.

Figure 6.8(b) shows an example of contraction, and the load balancing result is summarized in Table 6.7, Case 4.1. In Case 4.1, there are seven active nodes, but ten nodes have messages transmissions. The message details are listed
below:

BROADCAST: one message per active node for \{d, e, f, g, h, i, j\}.

DETECT: c → d → e → f → g → h → i → a → b → (c).

UPDATE: c → b → (a).

The traversal of contraction in the DETECT component is discussed in Section 5.4.2. And the traversal in the UPDATE component is discussed in Section 5.6 and illustrated in Figure 5.20(k). Therefore, the nodes \{b, c, d, e, f, g, h, i\} have two messages sent, and \{a, j\} have one message sent.

Generally, both expansion and contraction can achieve good load balancing. The maximum message load is three messages for expansion, and is only two messages for contraction. The messages have been well distributed to a set of nodes, and there is not a single node that has been overloaded.

### Merge and Split

Similar to expansion and contraction, merge and split also have good load balance, as shown in Table 6.7, Cases 5.1-5.4 and 6.1-6.4. But merge and split have more workloads than expansion and contraction. The maximum workload is four messages for merge, and three messages for split. The traversals of merge and split in the DETECT component is introduced in Section 5.4.3. The traversals in the UPDATE component is shown in Figure 5.21(b) and (e).

One of the examples of merge (Case 5.1) is shown in Figure 6.9(a), where there are three active nodes, i.e., c, f, and g. The message transmissions for the three components are as follows:

BROADCAST: one message per active node for \{c, f, g\}.

DETECT: b → c → d → e → f → g → a → (b).

DETECT: e → f → g → a → (b).

UPDATE: b → c → d → h → i → e → f → g → (a).

There are two traversals for merge in the DETECT component. One traversal is initialized and concluded by node b. The other traversal initialized by node e is terminated by node b, because both b and e are type 1 nodes and b has a shorter distance to the sink than e. As shown in Figure 6.9(a), nodes \{f, g\}, \{e, c\}, \{a, b, d\}, and \{h, i\} transmit four, three, two, and one messages respectively.

Figure 6.9(b) shows an example of split (Case 6.1), and the detail message transmission are as follows:

BROADCAST: one message per active node for \{d, g, h\}.

DETECT: c → d → e → f → g → h → a → b → (c).

DETECT: f → g → h → a → b → (c).

UPDATE: c → b → (a).
UPDATE: $f \rightarrow e \rightarrow i \rightarrow j \rightarrow k \rightarrow l \rightarrow (f)$.

The nodes $\{i, j, k, l\}$, $\{a, c, e, d\}$, and $\{b, f, g, h\}$ have sent one, two, and three messages respectively.

Self-merge and Partial-split

In the DETECT component, the traversals for self-merge and partial-split are similar to the traversals of merge and split, as illustrated in Figure 5.13(e)-(h). In the UPDATE component, the traversals for self-merge and partial-split are illustrated in Figure 5.21(h) and (k). Figure 6.10(a) and (b) are examples of self-merge and partial-split, and the results of these two examples are listed in Case 7.1 and Case 8.1 in Table 6.7.
Discussion

As a summary, all the eight types of changes have a good balance of workload in terms of messages transmissions. Messages have been well distributed to a group of sensor nodes, especially along traversals, so that no sensor nodes have been overloaded with messages. For example, self-merge spends the most messages in the UPDATE component among the eight types of changes as in Figure 6.5. But this large amount of messages has been distributed to a large number of nodes. For example, in Case 7.4, there are \(88 + 26 + 4 + 11 = 129\) nodes that have transmitted a total number of 196 messages.

Contraction has the best workload distribution among the eight types of changes with no sensor node transmitting more than two messages. Expansion, split, and partial-split have the second best workload balance with no more than three messages per node. Merge and self-merge follow with no more than four messages per node. The workload for appearance and disappearance can vary at different simulation runs, as already been discussed earlier in Section 6.2.3.

There are some special cases, where the load balance can be not as good as the cases in Table 6.7. Figure 6.11 shows an example of expansion. Generally,
the maximum workload for expansion is three messages per node as in Table 6.7, but in the example in Figure 6.11, the maximum workload is five messages for nodes $c$ and $d$. The detail messages are listed below:

**BROADCAST:** one message per active node for \{b, c, d, e, f\}.

**DETECT:** $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow d \rightarrow c \rightarrow f \rightarrow g \rightarrow (a)$.

**UPDATE:** $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow d \rightarrow c \rightarrow f \rightarrow (g)$.

Because nodes $c$ and $d$ are active nodes, both $c$ and $d$ have one messages sent in the BROADCAST component. In the DETECT component, both $c$ and $d$ send two messages (marked by underlines), because they have been visited by the traversal twice. The traversal in the UPDATE component also visits $c$ and $d$ twice. Thus, both $c$ and $d$ have totally sent five messages. The maximum messages sent in the special cases is only one or two messages more than the cases in Table 6.7, therefore the special cases should also have good load balance.
Chapter 7

Case Study

The objective of this chapter is to extend the previous chapter by providing more experiments. The experiments in the previous chapter use spatial objects with simple geometries. This chapter will use arbitrary complex spatial objects for the experiments. The spatial objects in this chapter are selected from a land use change dataset that consists of different land use types, such as forest, shrubbery, grassland, pond, desert, wasteland, and irrigated land. Each land use type has three snapshots from three different years. It is assumed that the time scale of the land use change dataset is appropriate for a sensor network deployment. And the main purpose of this chapter is to detect different types of changes in a realistic snapshot sequences using the decentralized change detection algorithm.

7.1 Study Area

Sangong river watershed is selected as the study area, since it is located within an area of dramatic land use changes [15]. As shown in Figure 7.1, the location of Sangong river watershed is in Xinjiang Uygur Autonomous Region in the Northwest of China. Sangong river watershed covers approximately 940 km2, and it extends in the south from the foot of the Tianshan Ranges, to the central oasis irrigation farmland, and further to the northern desert within Junggar Basin.

7.2 Experiment Results

In the first experiment in Figure 7.2(a)-(c), there are three time steps $t_1$, $t_2$, and $t_3$. The sensor network is tasked to detect changes that occur between $t_1$ and $t_2$ and between $t_2$ and $t_3$. At time step $t_2$, two expansions have been detected, i.e., $a \rightarrow \ldots \rightarrow b \rightarrow \ldots \rightarrow a$ and $c \rightarrow \ldots \rightarrow d \rightarrow \ldots \rightarrow e \rightarrow f \rightarrow e$ as in Figure 7.2(b).
At \( t_3 \), there are other two expansions \( g \rightarrow \ldots \rightarrow h \rightarrow \ldots \rightarrow i \rightarrow \ldots \rightarrow j \rightarrow \ldots \rightarrow g \) and \( i \rightarrow \ldots \rightarrow k \rightarrow \ldots \rightarrow l \rightarrow d \rightarrow \ldots \rightarrow i \) as in Figure 7.2(c).

The detail results for the first experiment are listed in the first row of Table 7.1. At \( t_2 \), there are 12 active nodes, in which three active nodes contribute to an expansion and the other nine active nodes belong to the other expansion, as in Figure 7.2(b). The average message sent per active node for time step \( t_2 \) is 3.67, which is higher than the average result of expansion (3.05) for 10 active nodes, as in Table 6.6. The reason is that the 12 active nodes in Figure 7.2(b) have been separated into two expansions of three and nine active nodes.

At time step \( t_3 \), there is an expansion of three active nodes and another expansion of 18 active nodes, as in Figure 7.2(c). The average messages for \( t_3 \), with 21 active nodes, is 3.95 that is even higher than the average messages of 3.67 at \( t_2 \) with 12 active nodes.

As discussed in Section 6.2.2, the message efficiency should be improved when the number of active nodes increases. For example, in Table 6.6, the average messages sent per active node is 3.05 with 10 active nodes, and the average messages are improved to 2.50 when the number of active nodes increases to 20.

The results of Case 1, Table 7.1 contradict the trend in Table 6.6, because the areal objects in Figure 7.2(c) (especially the changed area \( g \rightarrow \ldots \rightarrow h \rightarrow \ldots \rightarrow i \rightarrow \ldots \rightarrow j \rightarrow \ldots \rightarrow g \)) are long and thin. For example, if there are 100 nodes and they are arranged as a \( 10 \times 10 \) array, then a closed traversal along
Figure 7.2: Case 1. Inserted and deleted edges are marked as thick solid-line edges and thick dashed-line edges respectively. (b) At $t_2$, there are two closed trails, and two expansions have been detected. (c) Similarly, there are other two closed trails for expansions at $t_3$.

the boundary of the array will cost about $4 \times 10 = 40$ messages. In the other case, if the 100 nodes are organized as a $1 \times 100$ array, then the closed traversal will spend about $2 \times 100 = 200$ messages. Thus, generally, it is less efficient to traverse boundary edges of long and thin areal objects compared with square or circular areal objects.

In Case 2, expansions of an areal object have been detected at both time steps $t_2$ and $t_3$, as shown in Figure 7.3(b) and (c). At $t_2$, five expansions are detected by nodes $a$, $b$, $c$, $d$, and $e$ respectively. Each of these five nodes should receive a traversal that is initialized by itself. As discussed in Section 5.4.2, a traversal of expansion includes two segments. The first segment consists of inserted edges, and the second segment consists of deleted edges. In Figure 7.3(b) and (c), inserted and deleted edges are highlighted as thick solid-line edges and thick dashed-line edges respectively. A traversal initialized by node $c$, for instance, contain the segment of inserted edges $c \rightarrow \ldots \rightarrow g \rightarrow \ldots \rightarrow d$ and the segment of deleted edges $d \rightarrow \ldots \rightarrow c$. Similarly, at $t_3$, each of the four nodes $f$, $g$, $h$, and $i$ receives a closed traversal and detects an expansion.

The simulation result of Case 2 is listed in Table 7.1. The average messages sent per active node in Case 2 is less than that of Case 1, since generally there are more active nodes in Case 2. For example, in Figure 7.3(c), the expansion detected by node $i$ contains around 40 active nodes. But the result of Case 2
Table 7.1: Results of case study.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Time</th>
<th>Detected Changes</th>
<th>Active Nodes</th>
<th>BROADCAST</th>
<th>DETECT</th>
<th>UPDATE</th>
<th>TOTAL</th>
<th>Average Mgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>t₂</td>
<td>2 expansions</td>
<td>12</td>
<td>12</td>
<td>17</td>
<td>15</td>
<td>44</td>
<td>3.67</td>
</tr>
<tr>
<td></td>
<td>t₁</td>
<td>2 expansions</td>
<td>21</td>
<td>21</td>
<td>37</td>
<td>25</td>
<td>83</td>
<td>3.95</td>
</tr>
<tr>
<td>Case 2</td>
<td>t₂</td>
<td>5 expansions</td>
<td>39</td>
<td>39</td>
<td>48</td>
<td>35</td>
<td>122</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>t₁</td>
<td>4 expansions</td>
<td>61</td>
<td>61</td>
<td>59</td>
<td>44</td>
<td>164</td>
<td>2.69</td>
</tr>
<tr>
<td>Case 3</td>
<td>t₂</td>
<td>3 contractions</td>
<td>5</td>
<td>5</td>
<td>19</td>
<td>7</td>
<td>31</td>
<td>6.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 split</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁</td>
<td>4 contractions</td>
<td>17</td>
<td>17</td>
<td>34</td>
<td>19</td>
<td>70</td>
<td>4.12</td>
</tr>
<tr>
<td>Case 4</td>
<td>t₂</td>
<td>1 contraction</td>
<td>30</td>
<td>30</td>
<td>31</td>
<td>24</td>
<td>85</td>
<td>2.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 expansion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁</td>
<td>1 self-merge</td>
<td>5</td>
<td>5</td>
<td>21</td>
<td>88</td>
<td>114</td>
<td>22.80</td>
</tr>
<tr>
<td>Case 5</td>
<td>t₂</td>
<td>1 contraction</td>
<td>5</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>31</td>
<td>6.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 appearance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t₁</td>
<td>5 contractions</td>
<td>23</td>
<td>23</td>
<td>57</td>
<td>29</td>
<td>109</td>
<td>4.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 expansion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 partial-split</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 6</td>
<td>t₂</td>
<td>1 merge</td>
<td>36</td>
<td>36</td>
<td>46</td>
<td>39</td>
<td>121</td>
<td>3.36</td>
</tr>
</tbody>
</table>

is not as good as the result in Table 6.6, where the average messages per active node is only 2.16 for an expansion with 40 active nodes. One of the reasons is that there are multiple expansions in Case 2. In Figure 7.3(c), for example, there are four expansions with 6, 7, 8, and 40 active nodes (detected by \( f, g, h, \) and \( i \) respectively).

As listed in Table 7.1, there are three contractions and one split at time step \( t₂ \) in Case 3. In Figure 7.4(b), the three contractions are detected by nodes \( a, b, \) and \( d \), and the split is detected by node \( c \). For the split detected by \( c \), the first, third, and fourth segments in the DETECT component are \( c → ... → e, e → ... → f, \) and \( f → ... → c \) respectively, while the second segment is absent at node \( e \). Suppose the areal object \( A \) at \( t₁ \) has split into \( A \) and \( B \) at \( t₂ \), then at \( t₃ \) nodes \( g, h, i, \) and \( j \) detect contractions of \( A \), and node \( e \) detects the disappearance of areal object \( B \).

The average messages for Case 3 are 6.20 and 4.12 at time step \( t₂ \) and \( t₃ \) respectively, as in Table 7.1. Similar to Case 1 and 2, the message efficiency of Case 3 is not as good as the results in Table 6.6, due to the multiple local changes in Case 3.

In Case 4, node \( a \) detects a contraction and node \( b \) detects an expansion at time step \( t₂ \) as in Figure 7.5(b). The average message sent is 2.83. In Figure 7.5(c), node \( c \) detects that the areal object \( A \) at \( t₂ \) has self-merged into a region
Figure 7.3: Case 2. (b) At $t_2$, five expansions are detected by nodes $a$, $b$, $c$, $d$, and $e$ respectively. (c) $b$, $c$, $d$, and $e$ detects expansions at $t_3$. 
Figure 7.4: Case 3. (b) The sensor network detects three contractions and one split at $t_2$. (c) Four contractions and one disappearance are detected at $t_3$.

A with a hole $B^-$ at $t_3$. In the DETECT component at $t_3$, the first, second, and third segments of the traversal are $c \rightarrow \ldots \rightarrow f \rightarrow \ldots \rightarrow d$, $d \rightarrow e$, and $e \rightarrow \ldots \rightarrow f \rightarrow \ldots \rightarrow c$, and the fourth segment are absent at node $C$.

It can be seen in Case 4, Table 7.1 that the UPDATE component at $t_3$ costs 88 messages. As discussed in Section 5.6, if a self-merge is detected, two traversals should be initialized to determine the hole. In the example in Figure
Figure 7.5: Case 4. (b) Node $a$ detects a contraction, and node $b$ detects an expansion at $t_2$. (c) Node $c$ detects a self-merge at $t_3$.

7.5(c), there are two traversals for the UPDATE component: $e \rightarrow \ldots \rightarrow f \rightarrow \ldots \rightarrow c \rightarrow \ldots \rightarrow a \rightarrow \ldots \rightarrow b \rightarrow \ldots \rightarrow d \rightarrow e$ for the external boundary, and $c \rightarrow \ldots \rightarrow f \rightarrow \ldots \rightarrow d \rightarrow \ldots \rightarrow c$ for the internal boundary. The hole $B^-$ can then be determined based on area calculation. But the UPDATE component contributes to a result of 22.80 messages per active nodes, which is significantly less efficient than the result in Table 6.6 (row 7). The reason is that there is only five active nodes in Figure 7.5(c), and the size of the areal object is much larger in comparison with the number of active nodes. A lot of messages will be spent on the boundary traversals of the areal object.

As a result, in the case of self-merge, if the number of active nodes is constant, the increase of the size of an areal object will reduce the message efficiency of the decentralized change detection algorithm. It is worth noting that the algorithm will still have a good load balance in spite of the decrease of message efficiency. As discussed in Section 6.2.3, in the case of a self-merge, the large amount of messages will be distributed to a large number of nodes located at the boundary edges of the areal objects.

The snapshots for Case 5 is shown in Figure 7.6. At time step $t_2$, node $a$ detects a contraction of areal object $A$ with one active node, while node $b$ detects the appearance of a new hole $C^-$ with four active nodes. In Table 7.1, the average messages sent per active node for $t_2$, Case 5 is 6.20 that is much higher than the results in Table 6.6 (see row 1). The first reason is that there
Figure 7.6: Case 5. (b) At $t_2$, node $a$ detects a contraction of the areal object $A$, and node $b$ detects the appearance of a hole $C^-$. (c) Five contractions, one expansion, and one partial-split are detected at $t_3$. 
are smaller groups of active nodes, i.e., one and four, at $t_2$, Case 5. Secondly, there is an appearance of a hole $C^-$ at $t_2$, Case 5, as in Figure 7.6(b). On the other hand, the result of appearance in Table 6.6 is based on the appearance of areal objects.

Usually, the appearance of a hole is less message efficient than the appearance of an areal object. For example, in Figure 7.6(b), the appearance of the hole $C^-$ has only four active nodes, but the other ten non-active nodes, marked by circles, have to be involved in the change detection process. In detail, the BROADCAST component spends 4 messages for the four active nodes, the DETECT component spends 10 messages for a traversal along the boundary edges (referred to Figure 5.8(a)), and the UPDATE component spends 10 messages for another traversal along the boundary edges (referred to Figure 5.20(a)-(b)). The result is an average of $(4 + 10 + 10)/4 = 6$ messages per active node. In comparison, the appearance of an areal object is more efficient, e.g., 2.86 messages per active node for 10 active nodes in Table 6.6, because only active nodes have to be involved in detecting the appearance of an areal object.

At $t_3$ in Figure 7.6(c), five contractions are detected by nodes $c$, $d$, $e$, $f$, and $g$, and one expansion is detected by node $h$. Node $i$ detects a partial-split of areal object $A$ with nine active nodes, and the hole $B^-$ has disappeared. Different from self-merge, in the case of partial-split, the message efficiency will not be affected by the size of an areal object. However, the message efficiency of a partial-split will depend on the size of the hole, since the UPDATE traversal for a partial-split needs to visit the boundary edges of the hole, as illustrated in Figure 5.21(j)-(k). The size of the hole $B^-$ and the number of active nodes in Figure 7.6(b)-(c) are similar to the cases in Section 6.2 (e.g., Figure 6.10). Thus, the message efficiency for partial-split in Case 5, Table 7.1 should be close to the results in Table 6.6.

In the last case, there is a merge between the two consecutive snapshots in Figure 7.7(a)-(b). The four traversal segments in the DETECT component are: $a \rightarrow ... \rightarrow b \rightarrow ... \rightarrow c$, $c \rightarrow ... \rightarrow d \rightarrow ... \rightarrow e$, $e \rightarrow ... \rightarrow f \rightarrow ... \rightarrow g$, and $g \rightarrow ... \rightarrow h \rightarrow ... \rightarrow a$. As listed in Case 6, Table 7.1, there are 36 active nodes, and the average message per active node is 3.36, which is similar to the result of merge, i.e., 3.20 for 30 active nodes, in Table 6.6.

7.3 Discussion

The DETECT and UPDATE components in the decentralized change detection algorithm only consider a single change for each time step. However, most of the cases provided in this chapter consist of multiple changes, and thus there can be multiple traversals at each time step. For example, there are two expansions
Figure 7.7: Case 6. (b) Node $a$ detects a merge of two areal objects.

at $t_3$ in Case 1, therefore in the DETECT component two closed traversals are required to detect the two expansions respectively, and in the UPDATE component other two traversals are required to properly update the trajectory identity of the areal object.

The existing decentralized change detection algorithm presented in Chapter 5 is valid for certain coexistence of local changes. In Cases 1 and 2, for example, the algorithm works for multiple local expansions of an areal object. Because an expansion of an areal object will not cause the change of trajectory identity of the areal object, the coexistence of multiple expansions will also not change the trajectory identity. Thus, multiple local expansions do not need to be combined. Similar to expansions, multiple contractions of areal objects can also be processed locally without combinations.

Another example is in Case 3, where there are one split and three contractions of an areal object. Multiple contractions do not need to be combined with a split, so that the algorithm works for Case 3 as well. More generally, one merge/split/self-merge/partial-split and multiple expansions/contractions do not need to be combined. An expansion/contraction of an areal object does not affect the trajectory identity of the areal object, so an expansion/contraction can be simply processed locally using the decentralized change detection algorithm, regardless any other changes of areal objects. As a result, the algorithm works for one merge/split/self-merge/partial-split with multiple expansions/contractions. For the cases of two or more merge/split/self-merge/partial-split, the existing algorithm needs to be extended.

To detect the complex changes of spatial objects between two snapshots, future work is required to extend the decentralized change detection algorithm.
to process different combinations of traversals in both DETECT and UPDATE components, as will be discussed in Section 8.3. For example, in Figure 8.1(c)-(d), two merges can be detected locally, and the two local detections should be combined to obtain the global result, i.e., two areal objects merge into a region with a hole.

In Figure 7.4(a)-(b), one split is detected by node c, however, there is no split for the underlying spatial phenomenon (gray area). As already discussed in Section 3.3, if the granularity of observations for a sensor network is not appropriately set up for the environmental phenomenon, then errors of omission and errors of commission can be generated during the observation. In the example in Figure 7.4(a)-(b), the error is generated especially because of the imprecision of spatial data. Imprecision is a lack of detail in information, and it is an intrinsic feature of geographic information [28]. The real world phenomena are continuous, but the granular observation has discretized the phenomena spatially, temporally, and thematically, resulting in imprecision in spatiotemporal data. The evaluation of errors of omission and errors of commission is out of the scope of this thesis, and is a direction for future research.

7.4 Summary

In this chapter, six experiments have been provided. In each of these experiments, the spatial objects have arbitrarily complex geometries. As a result, the changes of these spatial objects among snapshot sequences can be much more complex than the simulated spatial objects in Chapter 6. Chapter 6 only considers a single change between two consecutive snapshots. While this chapter has illustrated that multiple changes can co-exist between two snapshots. The experiment results in Section 7.2 shows the potential of the decentralized change detection algorithm to process complex spatial objects. However, future work is required to extend the algorithm to deal with different combinations of spatial changes, as will be discussed in the next chapter.
Chapter 8

Conclusions

This chapter first has a summary of this thesis. The major contributions of this thesis are then presented. Finally, the ideas and suggestions for future research are presented.

8.1 Summary

The goal of this thesis is to bridge the gaps between theoretical event-based spatiotemporal data models and practical snapshot-based models, and this research has achieved the goal by introducing granularity into spatiotemporal data model. The data model proposed in Chapter 4 supports different spatial, temporal, and thematic granularity of observations. By using the data model in Chapter 4, the theoretical event-based approach is able to apply into practical applications.

With a certain temporal granularity of observation, the spatiotemporal data is organized as sequences of snapshots in the data model. Areal objects may undergo different types of changes between two consecutive snapshots, and the representation about the changes of areal objects has been investigated in the data model.

In a decentralized network, there are two categories of changes locally in a sensor node: one is the change of node status, and the other is the change of directed edges. The sensor readings have been classified into binary readings: either inside ($y_1$) or outside ($y_2$) of an areal object. For a node $v$, the change of readings between $y_1$ and $y_2$ represents that the node $v$ has locally detected some change of areal objects. However, the node $v$ locally may not know what type the change is, since between two consecutive snapshots with an arbitrary time scale, the change of an areal object can be gradual or abrupt. As defined in Chapter 4, gradual change is the change of one and only one node status,
while abrupt change involves the change of two or more node status.

To support granularity of observations, the data model in Chapter 4 should support both gradual and abrupt changes. In the case of abrupt changes, local data sets in the sensor nodes need to be aggregated, such that global knowledge, e.g., types of changes, can be derived from the aggregation results. In this circumstance, directed edges have been introduced, and the boundary of an areal object is represented as a sequence of directed edges. With the evolving of spatial objects over time, the sequence of directed edges may also change, therefore inserted and deleted edges are introduced to represent the change of directed edges over time.

As the introduction of inserted and deleted edges, the numbers of active nodes, i.e., the nodes that have changed readings, are allowed to be arbitrary, because inserted and deleted edges can be easily constructed by only one hello message (one-hop broadcast) per active node. Also, the inserted and deleted edges always chain together as closed traversable trails. For example, in the case of expansion of an areal object, the inserted and deleted edges will form a closed traversable trail, whenever the numbers of active nodes are 1, 10, 100, or 1000.

Based on the characteristics of inserted and deleted edges, a decentralized change detection algorithm has been proposed in Chapter 5, and it includes three components: BROADCAST, DETECT, and UPDATE. The BROADCAST component is responsible for transforming the change of node status into the change of directed edges, such that inserted and deleted edges can be determined. Regardless the number of active nodes, the DETECT component is then able to detect eight different types of changes based on the traversals of inserted and deleted edges, and the traversals in the DETECT component always form closed trails. Finally, the UPDATE component is responsible to properly update the trajectory identity of boundary edges.

For the reason of energy (messages) savings in sensor networks, transition edges have been defined, and the sensor nodes have been classified into three types, i.e., type 1, type 2, and type 3 nodes based on the definition of transition edges. Two traversal approaches are then proposed. Traversal Approach 1 is dependent to transition edges, and the traversals in this first approach are initialized and terminated by type 1 nodes. In contrast, the selection of the starting nodes in Traversal Approach 2 is not related to transition edges. Generally, Traversal Approach 1 is more efficient than Traversal Approach 2. Because there is limited number of type 1 nodes, the redundant traversals in the first approach can be avoided. On the other hand, starting nodes for traversals in the second approach is selected based on the rule of, for example, the shortest distance to the sink, so that redundant traversals is highly possible to be locally
The decentralized change detection algorithm in Chapter 5 has been applied in simulated sensor networks for both qualitative and quantitative reasonings. And the evaluation results in Chapters 6 and 7 proof the hypothesis that the new spatiotemporal data model proposed in Chapter 4 is capable for efficient qualitative and quantitative analysis in terms of messages costs. The evaluation also proof that the decentralized change detection algorithm is messages efficient, scalable, and has a good load balance among sensor nodes.

8.2 Contributions

This thesis has several contributions to geographical information science and decentralized spatial computing in sensor networks:

This thesis develops a new spatiotemporal data model that integrates both snapshot-based and event-based approaches for spatiotemporal data representation. The new data model supports spatial, temporal, and thematic granularity of observations, and represents spatial changes as sequences of snapshots. The model supports qualitative and quantitative analysis on snapshot sequences. Salient changes or events can be derived based on the analysis of snapshot sequences.

The primary data structure in the model is edges. Boundaries of areal objects are represented as edge sequences that provide the flexibility to support both gradual and abrupt changes. Spatial datasets acquired from real world applications usually consist of snapshot sequences where spatial change appears to be abrupt. The new data structure has the potential to be applied into practical applications for spatiotemporal data analysis due to its ability to handle both abrupt and gradual changes.

The new data structure, i.e., directed edges, ensures the boundaries of areal objects are always closed traversable trails. This thesis also defines deleted and inserted edges to represent changes of areal objects in snapshot sequences. The sequences of deleted and inserted edges always form closed traversable trails, and there are eight different types of these closed trails that can be used to distinguish six topological changes and two non-topological changes.

This thesis develops a decentralized change detection algorithm to efficiently detect the eight different types of changes in a sensor network. The algorithm maintains the boundary edges of areal objects in two consecutive time steps in a decentralized network, such that inserted and deleted edges can be calculated. The sequences of inserted and deleted edges can then be used to identity different type of spatial changes.

This thesis develops an approach to decentrally store the completed lifespan
of areal objects in a sensor network. The lifespan of an areal object is a set of snapshots that is linked together by the trajectory of the areal object. Each snapshot in a lifespan consists of sequences of boundary edges. The lifespan of areal objects can be further developed to support different types of spatial queries in a spatiotemporal database.

The experiments verify the efficiency and scalability of the decentralized change detection algorithm. And the experiment result support the important role of granularity in the design of spatiotemporal data model and decentralized algorithms. For example, there are more abrupt changes, i.e., more active nodes, between two consecutive snapshots when a coarser temporal granularity of observation is applied. And the experiments have demonstrated that the decentralized change detection algorithm is more efficient to detect lower-frequency abrupt changes than higher-frequency gradual changes. Also, the decentralized change detection algorithm is highly scalable when the number of active nodes increases. The experiment results suggest the appropriate design of granularity can significantly improve the efficiency of decentralized algorithms.

This thesis proposes a new hierarchical structure, called multi-granularity sensor network, to efficiently monitor the changes of environmental phenomena with different granularity of observations. There are two commonly-used network structures for in-network aggregation, i.e., peer-to-peer structure, and tree structure. This thesis has demonstrated the advantages of the proposed hierarchical network topology for monitoring environmental changes by comparison with other network topologies.

Additionally, this thesis proposes that the granularity of spatial changes can be defined, and the definition on the granularity of spatial changes should be related to the granularity of observations. The definition of granularity of observations provides a theoretical foundation for the future investigation on granularity of spatial changes.

8.3 Future Work

As discussed in the previous section, this thesis has made several major contributions. However, there are also various limitations in this thesis. These limitations could be addressed in the future work discussed below.

In this thesis, one of the limitations is that the decentralized change detection algorithm in Chapter 5 can only allow at most two areal objects, for instance, the merge of two areal objects or the split of one areal object into two areal objects. In Figure 5.13, there are eight different types of closed trails to identify different types of change, and each closed trails has at most four segments.

If there are more than two areal objects involved in a specific type of change,
then the closed trail for the change would consist of more than four segments. Figure 8.1(a) and (b) are two snapshots from a land use change dataset that represents the change of shrubbery over time. In Figure 8.1(a), there are three areal objects A, B, and C, and in Figure 8.1(b) the three areal objects have merged. Also, there is an expansion at A between Figure 8.1(a) and (b). The closed trail for the merge of A, B, and C will include six segments, i.e., \( a \rightarrow \cdots \rightarrow b, b \rightarrow \cdots \rightarrow c, c \rightarrow d, d \rightarrow \cdots \rightarrow e, e \rightarrow \cdots \rightarrow f, \) and \( f \rightarrow \cdots \rightarrow a \). Segments 1, 3, and 5 consist of inserted edges, while Segments 2, 4, and 6 consist of deleted edges. The decentralized change detection algorithm in Chapter 5 should be extended in the future work to process closed trails with more than four segments.

Another limitation of the decentralized change detection algorithm is shown in Figure 8.1(c)-(d). Eight different closed trails are introduced in Figure 5.13. In a spatial dataset about land use change, for instance, the combinations of the eight different closed trails will be in a large variety. The algorithm in Chapter 5 is not able to consider different types of combinations. As an example in Figure 8.1(c)-(d), two areal objects A and B have self-merged. There are two closed trails, i.e., \( a \rightarrow \cdots \rightarrow b \rightarrow \cdots \rightarrow c \rightarrow \cdots \rightarrow d \rightarrow \cdots \rightarrow a \) and \( e \rightarrow \cdots \rightarrow f \rightarrow \cdots \rightarrow g \rightarrow \cdots \rightarrow h \rightarrow \cdots \rightarrow e \). Suppose nodes \( a \) and \( e \) can receive and conclude the two closed trails, then \( a \) and \( e \) will locally detect a merge of A and B respectively. The local knowledge of nodes \( a \) and \( e \) needs to combine such that the global result, i.e., areal objects A and B merge into a region with a hole, can be obtained.

One solution is to consider Figure 8.1(c)-(d) as two changes, i.e., a merge (detect by, e.g., \( a \)), and later a self-merged (detect by, e.g., \( e \)). The algorithm can then consider the merge first, and in a later time step consider the self-merged. Since the algorithm is decentralized, the organization of the time in the decentralized network needs further investigation.

Spatial query in spatial databases, such as GIS, are often based on the relations among spatial objects [33]. For example, some typical spatial queries are: “find the names of all the countries that border Germany” or “find the names of all the rivers that flow through the state of Victoria”. Some of the possible relationships in Oracle Spatial Database, for example, include disjoint, touch, overlap, equal, contains, covers, inside, etc. One of the areas of future work is to design a decentralized algorithm for querying the relationships among arbitrary spatial objects, such as regions, lines, and points.

The data structure introduced in Chapter 4 can be used to determine the relationships among spatial objects. Figure 8.2(a) and (b) provide two examples. In Figure 8.2(a), two spatial objects A and B are disjoint. To determine the relationships of a spatial object A with other spatial objects, a traversal on the
Figure 8.1: Between (a) and (b), there is a merge of three areal objects A, B, and C, and an expansion at areal object A.
boundary edges of the spatial object \( A \) is required. Suppose the traversal for the spatial object \( A \) starts at node \( a \), then the traversal would be \( a \rightarrow b \rightarrow \cdots \rightarrow c \rightarrow a \). Local knowledge of the nodes on the boundary edges will be aggregated during the traversal. The local knowledge of a node \( v \) includes the dataset of the node \( v \) itself and the datasets of its immediate neighbors. For example, node \( a \) in Figure 8.2(a) should aware four edges \((c, a), (a, b), (f, d), \) and \((d, e)\).

Figure 8.2(b) illustrates an overlap of two spatial objects \( A \) and \( B \). The boundary edges of \( A \) and \( B \) are two closed trails, i.e., \( a \rightarrow b \rightarrow \cdots \rightarrow c \rightarrow d \rightarrow a \) and \( a \rightarrow e \rightarrow d \rightarrow f \rightarrow \cdots \rightarrow g \rightarrow a \) respectively. Notice that in Figure 8.2(b) node \( a \) has four boundary edges \((d, a), (a, b), (g, a), \) and \((a, e)\), in which two of the edges \((d, a)\) and \((a, b)\) belong to object \( A \) and the other two edges \((g, a)\), and \((a, e)\) belong to object \( B \). Similarly, node \( d \) also has four boundary edges. The local datasets needs to be aggregated to determine the relations among spatial objects.

The relationships among line objects and areal objects can also be deter-
mined as shown in Figure 8.2(c) and (d). In Figure 8.2(c), there is a line object A and an areal object B. In the proposed data structure in Chapter 4, the line object can be regarded as a special type of areal object, since the edges of a line object also form a closed trail. For example, the edge sequence of the line A in Figure 8.2(c) is: $a \rightarrow d \rightarrow a \rightarrow e \rightarrow f \rightarrow e \rightarrow a$, and the line A intersects with the region B. Notice that node a has six boundary edges, where four edges $(e, a)$, $(a, d)$, $(d, a)$, and $(a, e)$ belong to the line A and two edges $(c, a)$ and $(a, b)$ belong to the region B. Future work is required to design a decentralized algorithm for the aggregation of local datasets along the traversal of boundary edges, so that relationships among spatial objects can be determined in a decentralized network.

Based on the static relationships among spatial objects, changes in topological relationships can then be further investigated. Changes in topological relations among spatial objects can occur when spatial objects undergo different types of spatial changes, e.g., expansion, contraction, merge, and split.

Figure 8.3 provides two examples. In the first example, an areal object A expands between time steps $t_1$ and $t_2$ in Figure 8.3(a)-(b), while areal object B remains unchanged. The detection of changes in topological relations between time steps $t_1$ and $t_2$ should be based on the condition that the static relationships among spatial objects are already available at $t_1$. For example, at time step $t_1$ in Figure 8.3(a), both areal objects A and B should know that they do not have any interaction with any other spatial objects.

As discussed in Section 5.4.2, there is a traversal for the expansion of areal object A, and the traversal will start at node a as in Figure 8.3(b). The traversal initialized at a will visit nodes b and c and finally return to a, i.e., $a \rightarrow g \rightarrow b \rightarrow c \rightarrow \ldots \rightarrow d \rightarrow \ldots \rightarrow a$. To detect the changes of topological relations, local datasets should be aggregated along the traversal. In Figure 8.3(b), two important local datasets are located at nodes b and c. In node b, there are two inserted edges of object A, i.e., $(g, b)$ and $(b, c)$, and importantly two boundary edges of object B, i.e., $(c, b)$ and $(b, e)$. Similarly, node c has two edges from A and two edges from B. When the traversal for expansion returns to its original node a, a should be able to detect the change of topological relations between areal objects A and B, i.e., A and B are disjoint at time $t_1$ but they become overlap at $t_2$.

Another example is shown in Figure 8.3(c)-(d), where two areal objects A and B merge into one areal object between $t_1$ and $t_2$, and the line object C does not change. If the traversal for the merge of A and B starts and concludes at node a, then the traversal would be: $a \rightarrow \ldots \rightarrow e \rightarrow b \rightarrow \ldots \rightarrow c \rightarrow f \rightarrow d \rightarrow \ldots \rightarrow a$ (see Section 5.4.3 for more details about merge). The traversal for merge will aggregate local datasets from two important nodes e and f. Node
Figure 8.3: Changes of topological relationships when spatial objects undergo different types of changes. (a)-(b) Areal object $A$ has expanded (c)-(d) Areal objects $A$ and $B$ has merged into one areal object.

e has four boundary edges from the line object $C$ and two inserted edges, as shown in Figure 8.3(d). Similarly, node $f$ also has six edges. After the traversal is concluded at node $a$, $a$ would be able to detect the change of relations among spatial objects $A$, $B$, and $C$, based on the aggregated local information along the traversal. A detailed decentralized algorithm is needed to ensure the changes in topological relations are correctly detected over time.

The related work in geosensor networks usually assumes that a WSN is homogeneous. However, it is not a strict requirement that all nodes in a sensor network are homogenous. In practice a sensor network can be heterogeneous and include different types of sensor nodes with different capabilities. Heterogeneous sensor networks should be a good direction for future works. For example, some of the sensors may have shorter communication range and can sense temperature and moisture. While other sensors may have longer communication range and can sense light. Conceptual models, data structures, and decentralized algorithms need to be further designed for heterogeneous sensor
networks.

Furthermore, an interesting study would be to further develop the data model in Chapter 4 for the integration of heterogeneous datasets. For example, suppose that there are two spatial datasets $X$ and $Y$. The two datasets may have the similar taxonomy, e.g., shrub land, agricultural, forest, desert, and hydrology. But the two datasets can come from two different data sources, so that the geometries, e.g., polygons and lines, in these two datasets are not the same. The model can be further extended for the decentralized analysis of these different geometries between the two datasets $X$ and $Y$, so that a new dataset, e.g., $Z$, that has higher data quality and more information in term of geometry and taxonomy can be efficiently generated in a decentralized manner.
Bibliography


Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
SHI, MINGZHENG

Title:
Detecting change in an environment with wireless sensor networks

Date:
2010

Citation:

Persistent Link:
http://hdl.handle.net/11343/35786

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.